НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Система безпеки розумного дому з використанням web-socket (серверна частина)»

Виконав:
студент IV курсу, групи ІО-63
Лахуров Артем Ігорович _____

Керівник:
асистент кафедри ОТ
Стешин Віктор Васильович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.
Сімоненко Валерій Павлович _____

Рецензент:

_____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Лахурову Артему Ігоровичу

1. Тема проєкту «Система безпеки розумного дому з використанням web-socket», керівник проєкту Стешин Віктор Васильович, асистент кафедри ОТ, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту див. технічне завдання

4. Зміст пояснювальної записки дослідження предметної області, огляд існуючих рішень, визначення вимог і завдань для програмного продукту, вибір платформи та технології, обґрунтування оптимальності використання обраних інструментів для розробки, реалізація проєкту.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В.П.		

7. Дата видачі завдання _____

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломного проєкту.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Відмітки про виконання
1	*Затвердження теми роботи*	01.09.2019	виконано
2	*Вивчення та аналіз завдання*	02.09.2019-02.02.2020	виконано
3	*Розробка архітектури та загальної структури систем*	03.02.2020-03.03.2020	виконано
4	*Розробка структур окремих Підсистем*	04.03.2020-15.03.2020	виконано
5	*Програмна реалізація системи*	16.03.2020-12.04.2020	виконано
6	*Оформлення пояснювальної записки*	13.04.2020-17.05.2020	виконано
7	*Захист програмного продукту*		
8	*Передзахист*	26.05.2020	
9	*Захист*		

Студент Артем ЛАХУРОВ

Керівник Віктор СТЕШИН

Анотація

В бакалаврській дипломній роботі реалізована серверна частина для веб-додатку системи безпеки розумного дому з використанням web socket. Цей додаток допомагає користувачам керувати пристроями розумного дому, а також отримувати детальну інформацію про пристрої. Програмний продукт був реалізований на мові Javascript за допомогою програмної платформи Node.js а також фреймворку Express.

Annotation

In the Bachelor's Degree the server part for the smart home security system web application using web socket is implemented. This app helps users to manage smart home devices as well as get detailed information about the devices. The software was implemented in Javascript using the Node.js software platform as well as the Express framework.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.467100.002 ТЗ	Система безпеки розумного дому з використанням web-socket (серверна частина). Технічне завдання	4	
3	А4	ІАЛЦ.467100.003 ПЗ	Система безпеки розумного дому з використанням web-socket (серверна частина). Пояснювальна записка	60	
4	А4	ІАЛЦ.467100.004 А1	Система безпеки розумного дому з використанням web-socket (серверна частина). Схема структурна.	1	
5	А4	ІАЛЦ.467100.005 А2	Система безпеки розумного дому з використанням web-socket (серверна частина). Схема функціональна.	1	
6	А4	ІАЛЦ.467100.006 А3	Система безпеки розумного дому з використанням web-socket (серверна частина). Схема принципова.	1	

Технічне завдання
до дипломного проєкту

на тему: «Система безпеки розумного дому з використанням web-socket (серверна частина)»

Київ – 2020

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку системи безпеки розумного дому з використанням web socket (серверна частина).

Область застосування: Система безпеки розумний дім може використовуватися в приватних будівлях, квартирах, а також в нежилих будівлях.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить завдання на виконання розробки серверної частини системи безпеки розумного дому з використанням web-socket, затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний Інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка програми системи безпеки розумного дому з використанням web socket(серверна частина).

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки служать науково-технічна література з комп’ютерних технологій, публікації в періодичних виданнях, публікації в Інтернеті за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- Розробка API для входу та реєстрації користувачів;
- Розробка API для керування пристроями;
- Розробка API для додавання нових пристроїв.

5.2. Вимоги до програмного програмного забезпечення

- Операційна система Windows, Linux, macOS
- Node.js 12.16.3

6. ЕТАПИ РОЗРОБКИ

	Дата
Затвердження теми роботи	01.09.2019
Вивчення та аналіз завдання	02.09.2019-02.02.2020
Розробка архітектури та загальної структури систем	03.02.2020-03.03.2020
Розробка структур окремих підсистем	04.03.2020-15.03.2020
Програмна реалізація системи	16.03.2020-12.04.2020
Оформлення пояснювальної записки	13.04.2020-17.05.2020
Захист програмного продукту	
Передзахист	26.05.2020
Захист	

Пояснювальна записка
до дипломного проєкту

на тему: Система безпеки розумного дому з використанням web socket (серверна частина)

Київ - 2020 року

ЗМІСТ

ВСТУП

Сьогодні набирають популярність системи безпеки розумного дому. Такі системи забезпечують безпеку дому та захищають від надзвичайних ситуацій. До їх складу входить охоронно-пожежна сигналізація, відеоспостереження всередині дому, відеодомофон, охорона периметру і т.п.

Актуальність теми

Через зростання популярності використання систем безпеки розумного дому, виникає потреба у розробці програм, що дозволять простіше та швидше налаштувати таку систему.

Мета і задачі дослідження

Метою роботи є розробка системи безпеки розумного дому з використанням web-socket , що дозволить простіше та швидше налаштовувати таку систему.

Для досягнення поставленої мети були поставлені наступні основні задачі:

- Провести аналіз існуючих систем безпеки розумного дому;
- Створити програмну реалізацію розробленої системи;
- Провести тестування розробленої системи.

Практичне значення

Запропонована система безпеки розумного дому полягає у простішому та швидшому часі налаштування такої системи за рахунок використання web-socket, яка дозволяє встановлювати з’єднання між клієнтом та сервером для обміну повідомленнями у режимі реального часу. Це дозволить швидше налаштовувати таку систему.

РОЗДІЛ 1
ОГЛЯД СИСТЕМ БЕЗПЕКИ РОЗУМНОГО ДОМУ

"Розумний дім"- це система управління комфортом і безпекою будинку. Системи безпеки потрібні для забезпечення безпеки будинку, захиститу від надзвичайних ситуацій. Сюди входять: захист від вторгнення за допомогою камер відеоспостереження, автоматизації дверей, воріт, охоронної сигналізації, запобігання аварійним ситуаціям.

Системи безпеки забезпечують:

- Контроль цілісності периметра (двері і вікна);
- Автоматизований контроль доступу в приміщення;
- Відеоспостереження за прилеглою територією;
- Отримання картинки з будь-якої камери відеоспостереження через Інтернет;
- Автоматичне освітлення території при проникненні.

1.1. Різновиди систем розумного дому

Системи розумного дому поділяються на:

- Дротові;
- Бездротові;
- Централізовані;
- Децентралізовані;
- З відкритим протоколом;
- З закритим протоколом.

1.1.1. Дротова система

У дротовій системі всі керуючі пристрої - датчики, вимикачі, пристрої управління кліматом, різноманітні керуючі панелі зв'язуються єдиною дротовою інформаційною шиною, якою йдуть сигнали до виконавчих пристроїв.

Переваги дротової системи:

- Надійність;
- Швидкість відгуку;
- Різноманітність інтегрованих систем;
- Довгий термін служби;
- Пожежна безпека.

1.1.2. Бездротова система

У бездротових системах, на відміну від дротових, сигнал від керуючих пристроїв до виконавчих йде по радіоканалу, а не по дротах. Це дозволяє скоротити кількість дротів, а також час на інсталяцію системи. Ці системи можна монтувати на об'єкти з готовим ремонтом з класичною проводкою. Кожен бездротовий "вимикач" є ще і радіопередавачем, який зв'язується з усіма іншими "вимикачами". Це дозволяє створювати різні світлові сценарії (нічний режим, вимкнути усе і т.п.) та перепрограмовувати функціонал клавіш.

Переваги бездротової системи:

- Можна встановлювати в квартири і будинки з уже готовим ремонтом з класичною проводкою;
- Зменшення кількості дротів, у порівнянні з дротовою системою;
- На відміну від дротової системи, проектування не потребується;
- Вартість.

1.1.3. Централізована система

Суть централізованого розумного дому полягає в тому, що програмується один центральний логічний модуль. Зазвичай це вільно програмований контролер з великою кількістю виходів. У контролер записується заздалегідь спеціально створена під об'єкт програма, на основі якої йде управління виконавчими пристроями та інженерними системами. Це

дозволяє використовувати широкий вибір обладнання та складних сценаріїв. Централізовані системи можуть бути як дротовими, так і бездротовими.

Переваги централізованої системи:

- Можливість управління всіма інженерним системами в єдиному інтерфейсі;
- Можливість створювати складні сценарії, прив'язані до часу доби, станом мешканця, температурі, місячним циклом;
- Можливість підключення майже будь-якого обладнання.

1.1.4. Децентралізована система

У децентралізованих системах "Розумного дому" кожен виконавчий пристрій несе в собі мікропроцесор з енергонезалежною пам'яттю. Цим пояснюється надійність таких систем. При виході з ладу одного пристрою вся система працює справно, крім приладів підключених до цього пристрою. Прикладом децентралізованої системи є "розумний дім" побудовані на основі протоколу KNX (найпопулярнішого в Європі).

Переваги децентралізованої системи:

- Надійність. Усі пристрої не залежать один від одного і мають енергонезалежну пам'ять.
- Популярність;
- Можливість використовувати додатковий блок логіки, який буде відповідати за специфічні сценарії;

1.1.5. Системи автоматизації з відкритим протоколом

Протокол - це мова якою спілкуються всі пристрої в "розумному домі". Якщо взяти протокол KNX, то він є відкритим. Багато виробників виготовляють пристрої, що працюють на цій мові. Асоціація KNX перевіряє їх на сумісність і тестує. Логотип KNX EIB на пристрої гарантує підвищену якість.

Переваги системи з відкритим протоколом:

- Великий вибір виробників. Це означає, що є великий вибір пристроїв за дизайном, ціною, характеристиками;
- Оновлення та конкуренція. Виробники конкурують в одному сегменті, що змушує їх розвиватися і винаходити нові пристрої.

1.1.6. Системи автоматизації з закритим протоколом

Для того, щоб спростити процес програмування, зменшити витрати на виробництво, деякі виробники випускають обладнання, яке працює на власному закритому протоколі. Крім них ніхто таке обладнання не випускає.

Переваги системи з закритим протоколом:

- Наявність цікавих рішень за нижчою ціною;
- Нижча вартість, ніж у систем з відкритим протоколом;
- Більш швидке реагування на вимоги ринку.

1.2. Існуючі системи розумного дому

1.2.1. Система розумного дому Ajax

Дана система автоматизації будинку в повній мірі справляється відразу з двома важливими завданнями: забезпечує комфорт і зручність в управлінні життєзабезпеченням приміщення; гарантує безпеку житла в повній мірі, контролюючи кордон об'єкта на предмет злому, а також електричну, пожежну, газову та інші можливі загрози для будинку.

Обладнання «Розумний дім» Ajax працює на надійно зашифрованому і захищеному двосторонньому радіозв'язку Jeweller власної розробки, має повну автономність від електромережі завдяки резервному джерелу живлення. На рис.1.1 зображено користувацький інтерфейс системи Ajax.

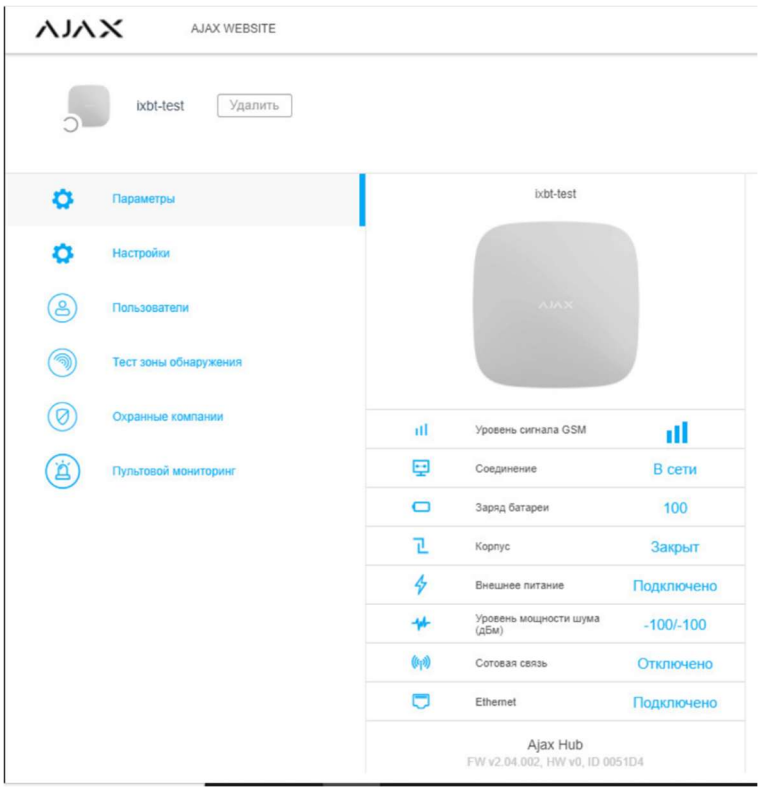


Рис. 1.1. Користувацький інтерфейс системи Ajax

Переваги системи:

- Простий монтаж;
- Бездротовий канал зв'язку між системними елементами;
- Велика зона дії сигналу (до 2000 м);
- Автономна робота хабу від акумулятора (до 16 годин);
- Wi-Fi і GSM-зв'язок;
- Різноманітність способів інформування користувача (дзвінок, SMS, Push-повідомлення);

Недоліки системи:

- Функціонування тільки з роботою центрального контролера (Hub), тобто відсутність автономності датчиків;
- Немає власної камери відеоспостереження (але є можливість підключення стороннього обладнання);
- Управління тільки через телефон.

1.2.2. Система розумного дому BroadLink

Обладнання «Розумний дім» BroadLink є комплектом сучасних цифрових пристроїв, створених для раціонального управління побутовою технікою, а також освітлювальною, енергетичною, охоронною та іншими системами в будинку. Кожен елемент такого комплексу може працювати як самостійно, так і взаємодіяти один з одним.

Переваги системи:

- Швидко встановлюється, підключається і настроюється;
- Функціонує без центрального хаба (автономна робота датчиків);
- Бездротова взаємодія пристроїв між собою;
- Контролюється по Wi-Fi.

Недоліки системи:

- Невелика дальність дії сигналу (до 50 м);
- Відсутність резервного живлення хаба.

1.2.3. Система розумного дому Fibaro

Розумний будинок Fibaro відноситься до професійного обладнання із забезпечення автоматизації та безпеки будинку з найширшим функціоналом. Однак, на відміну від багатьох подібних систем, потребує встановлення та налаштування своєї апаратури досвідченими фахівцями. На рис. 1.2 зображено інтерфейс системи Fibaro.

Рис. 1.2. Інтерфейс системи Fibaro

Переваги системи:

- Великий вибір сценаріїв для користувача;
- Розсилка повідомлень відразу на кілька телефонів;
- Робота на базі протоколу Z-Wave, що дозволяє успішно взаємодіяти з іншим подібним обладнанням;
- Невелика дальність сигналу системи збільшується за рахунок можливості кожного її елемента бути ретранслятором сигналу;
- Голосове управління через сервіси Google, але тільки англійською мовою.

Недоліки системи:

- Тільки професійний монтаж і налагодження;
- Обов'язкове підключення центрального контролера Fibaro Home Center до інтернету через LAN-кабель;
- Неможливість функціонування без центрального хаба;
- Відсутність резервного живлення хаба.

1.2.4. Система розумного дому Orvibo

Orvibo - це недорогий комплект простого в експлуатації обладнання, головне завдання якого полягає забезпеченні безпеки будинку. І тільки в другу чергу така установка може служити базою для організації повноцінної системи розумного дому.

Переваги системи:

- Простота в установці і підключенні;
- Віддалений контроль через додаток на смартфоні;
- Автоматичне знаходження та підключення сенсорів до центрального хабу;
- Широкий вибір пристроїв і можливість масштабування системи (близько 100 датчиків), причому інших виробників;
- Бездротовий протокол взаємодії між контролером і датчиками (ZigBee);
- Автономність деяких пристроїв від центрального хаба;
- Вибір сценаріїв роботи з технікою дому.

Недоліки системи:

- Невелика зона дії сигналу (до 30 м);
- Відсутність резервного живлення хаба на випадок відключення електроенергії;
- Дротове підключення до Інтернету (для надійності роботи системи).

1.2.5. Система розумного дому Xiaomi

Розумний дім від Xiaomi відноситься до бюджетного класу обладнання, яке дозволяє зробити управління різними пристроями і побутовою технікою в будинку максимально простим і зручним. Таким чином, комплект Xiaomi є відмінною стартовою платформою для підключення інших сенсорів і пристроїв, в тому числі сторонніх виробників. Її елементи працюють як

самостійно, так і в якості єдиного цілого. З їх допомогою можна створити досить функціональну систему контролю житлового простору, включаючи забезпечення безпеки. На рис. 1.3. зображено Android-додаток Xiaomi.

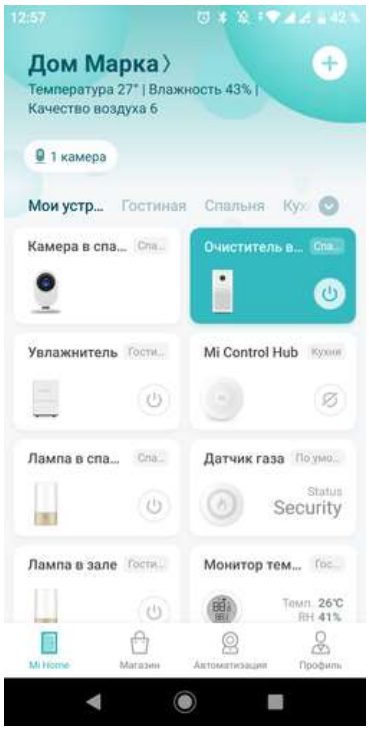


Рис. 1.2. Android-додаток Xiaomi

Переваги системи:

- Повна автономність пристроїв;
- Можливість масштабування;
- Бездротовий протокол ZigBee;
- Зручне управління за допомогою смартфона через Wi-Fi;
- Наявність настроюваних сценаріїв.

Недоліки системи:

- Мала зона дії сигналу (до 10 м);
- Відсутність резервного живлення хаба.

ВИСНОВКИ ДО РОЗДІЛУ 1

1. У даному розділі було проаналізовано системи безпеки розумного дому.
2. Було проаналізовано різновиди систем розумного дому, їх переваги та недоліки.
3. Було проаналізовані існуючі системи розумного дому, їх переваги та недолік

РОЗДІЛ 2.
ПРОЕКТУВАННЯ ДОДАТКУ

2.1. Опис предметної області

Система безпеки розумного дому це єдина система управління та контр оля комфортом та безпекою приміщення. Головною ціллю цієї системи є забе зпечення безпеки та комфортного користування всіма пристроями, які приєд нані до цієї системи.

До цілей входять: автоматизація світла, автоматизація дверей, запобіга ння аварійних ситуацій, захист від вторгнення, тощо.

2.1.1. Технологія Websocket

Протокол Websocket дозволяє обмін данних між клієнтською та сервер ною частиною через безперервне з'єднання. За допомогою цього данні можли во передавати в обох напрямках у вигляді пакетів, без додаткових HTTP-запитів та без розриву з’єднанння. Принцип роботи протоколу websocket показано на рис. 2.1.

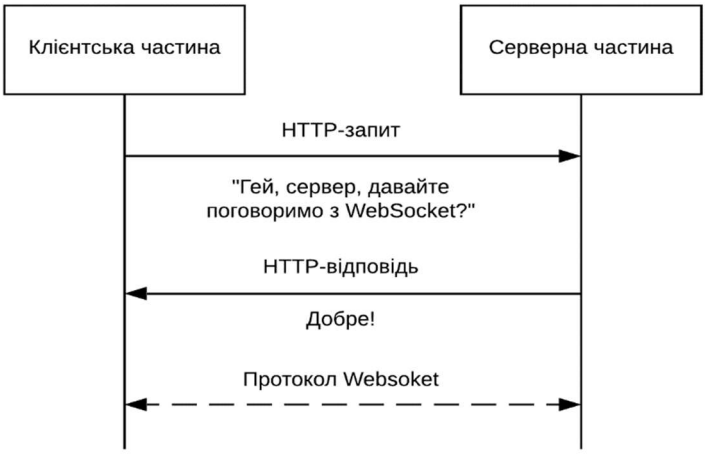


Рис. 2.1 Схема протоколу websocket

2.1.2. Node.js

Node.js – програмна платформа, заснована на движку V8 (здійснює тра нспіляцію JavaScript в машинний код), що трансформує JavaScript з вузькосп еціалізованої мови в мову загального призначення.

Node.js надає змогу JavaScript взаємодіяти з пристроями вводу-виводу через свій API (написаний на C++) підключаючи інші зовнішні бібліотеки, написані на різноманітних мовах програмування, забезпечуючи виклики до них з JavaScript-коду.

Для маніпулювання модулями можливо використовувати пакетний мен еджер npm (node package manager). Засновником платформи є Раян Дал.

Node.js має такі властивості:

- Асинхронна однониткова модель виконання запитів;
- Неблокуючий ввід/вивід
- Система модулів CommonJS
- Рушій JavaScript Google V8

Переваги Node.js:

- Неблокуюча модель введення-виведення.
  Ця система керована подіями і працює асинхронно, вибудовуючи чергу за пріоритетністю. Коли до сервера одночасно підключаються тисячі людей, їй легше впоратися з навантаженням, так як немає потреби створювати окремий потік для кожного підключення.
  Грамотний розподіл ресурсів допомагає витримати більшу кількість з'єднань. Тому така модель ідеальна для того, щоб створити чат бота або використовувати її в онлайн-грі.
- Можливість застосовувати одну мову на клієнті і сервері. Якщо програміст прокачався в JavaScript, йому буде легше вивчити "надбудову", ніж кардинально відрізняється технологію.

- Загальний код. Коду, використовуваного і на клієнті, і на стороні сервера, трохи, але він присутній. Головне - розуміти, що часто об'єкти з однаковими назвами можуть виконувати абсолютно різні функції в браузері і на серверній частині. Розроблявся спеціально для web. Вільно взаємодіє з найбільш популярними базами даних, допомагає отримати низькорівневий доступ (http, udp, https, tcp).
- Швидкість. Створення робочого, який впораэться з навантаженням, прототипу не відніме багато часу. Перший етап, коли програміст формує кістяк майбутнього продукту, проходить дуже швидко. Якщо архітектура добре продумана, то в подальшому труднощів з тим, щоб розширювати сайт на Node JS теж не з'явиться.
- Велике і бадьоре ком'юніті. Так як відкритий код, веб-розробники можуть писати різні модулі та пакети і ділитися ними. Найчастіше, модулі добре комбінуються.
- Синтаксис JavaScript. Так, доведеться вивчити нові інструменти, але буде багато звичного. Це забезпечує відносну простоту і доступність. Вбудована бібліотека. З кожним оновленням вона розширюється, але навіть в першій версії були досить широкі можливості. Постійний розвиток екосистеми. Кількість готових модулів і зовнішніх бібліотек безперервно зростає, в чому сприяє використання npm.

Недоліки Node.js:

- Гнучкість і швидкий розвиток породжує також і мінуси тому треба постійно стежити за оновленнями, деякі речі виходять недостатньо протестованими.
- Якщо перший розробник, який розробив модуль, видалить модуль з пакетного менеджеру, який використовує другий розробник для свого проекту, то проект другого розробника не запрацює.

2.1.3. V8

V8 - це високоефективний механізм JavaScript і WebAssembly з відкритим кодом Google, написаний на C ++. Він використовується в Chrome та Node.js, серед інших. Розробка JavaScript-движка V8 почалася в датському відділенні Google в місті Орхусі. Провідним розробником став Ларс Бак (Lars Bak). Головними проблемами, які довелося вирішувати розробникам цього движку, стали продуктивність і масштабованість. Перша лабораторна версія движка з'явилася 3 липня 2008 року, а вже 2 вересня була офіційно представлена версія 0.2.5, яка увійшла в перший публічний реліз Chromium.

Переваги V8:

- Компіляція вихідного коду JavaScript безпосередньо в власний машинний код, минаючи стадію проміжного байт-коду.
- Ефективна система управління пам'яттю, яка веде до швидкого об'єктного виділення і маленьким паузам збірки «сміття»
    a. V8 зупиняє виконання коду під час виконання збірки «сміття».
    b. Зменшує вплив припинення застосування при складанні «сміття».
    c. V8 може точно визначати, де знаходяться в пам'яті об'єкти і покажчики, що дозволяє уникнути витоку пам'яті при помилкової ідентифікації об'єктів в якості покажчиків.
- Введення прихованих класів і вбудованих кешей, що прискорюють доступ до властивостей і викликам функцій.

V8 виконує JavaScript-сценарії в особливих «контекстах», які по суті є окремими віртуальними машинами. Правда в одному процесі може працювати тільки одна віртуальна машина, незважаючи на можливість використання декількох потоків. У Chromium це обходиться мультипроцесорною архітектурою, що підвищує також стабільність і безпеку, реалізуючи таким чином механізм «пісочниці».

Таким чином, незважаючи на динамічну природу JavaScript, розробникам вдалося застосувати методи, характерні для реалізації класичних об'єктно-орієнтованих мов, такі як компіляція коду «на льоту», внутрішнє кешування, точний процес складання сміття, снепшоттінг при створенні контекстів.

Є передумови того, що в найближчому майбутньому для V8 можна буде використовувати не тільки JS але і TypeScript або інші. На даний момент йде транспайлінг цих мов в JS. У майбутньому, вони ймовірно будуть підтримуватися з коробки, і все буде працювати набагато швидше.

2.1.4. CommonJS

CommonJS - це добровільна робоча група, яка проектує, прототипує і стандартизує різні JavaScript API. На сьогоднішній день вони ратифікували стандарти для модулів і пакетів - CommonJS визначають простий API для написання модулів, які можуть бути використані в браузері за допомогою тега <script>, як з синхронної, так і з асинхронної завантаженням. У структурному плані, CommonJS-модуль являє собою готовий до повторного користування фрагмент JavaScript-коду, який експортує спеціальні об'єкти, доступні для використання в будь-якому залежному коді.

2.1.5. Порівняння Node.js та Java

Java - це мова, Node.js - можна назвати екосистемою побудованої на базі JS, і, перш за все, на базі V8.

Однак, коли ми говоримо про Java, ми говоримо не тільки про мову, а про віртуальну машину Java, а також всю екосистему і інфраструктуру побудовану навколо цієї машини. Як мінімум, їх можна порівнювати за цією ознакою - як результат, в обох випадках, ми маємо середу виконання. У разі Java - це віртуальна машина. У разі node.js - це двигун V8 який представлений на більшості ОС, таких як Windows, Linux, MacOS і менш відомих.

Розробники можуть писати код користуючись однією і тою ж мовою, і це буде працювати більш-менш однаковим чином на різних ОС за рахунок того, що існує середовище виконання. Середовище виконання впливає на те, як відбувається взаємодія з ОС. Крім того, їх можна порівнювати тому вони використовуються для вирішення схожого кола завдань.

Перш за все варто відзначити, що продуктивність Java набагато вище ніж у JS, і, відповідно, node.js. На рисунку 2.2 продемонстровано продуктивність Node.js та Java.

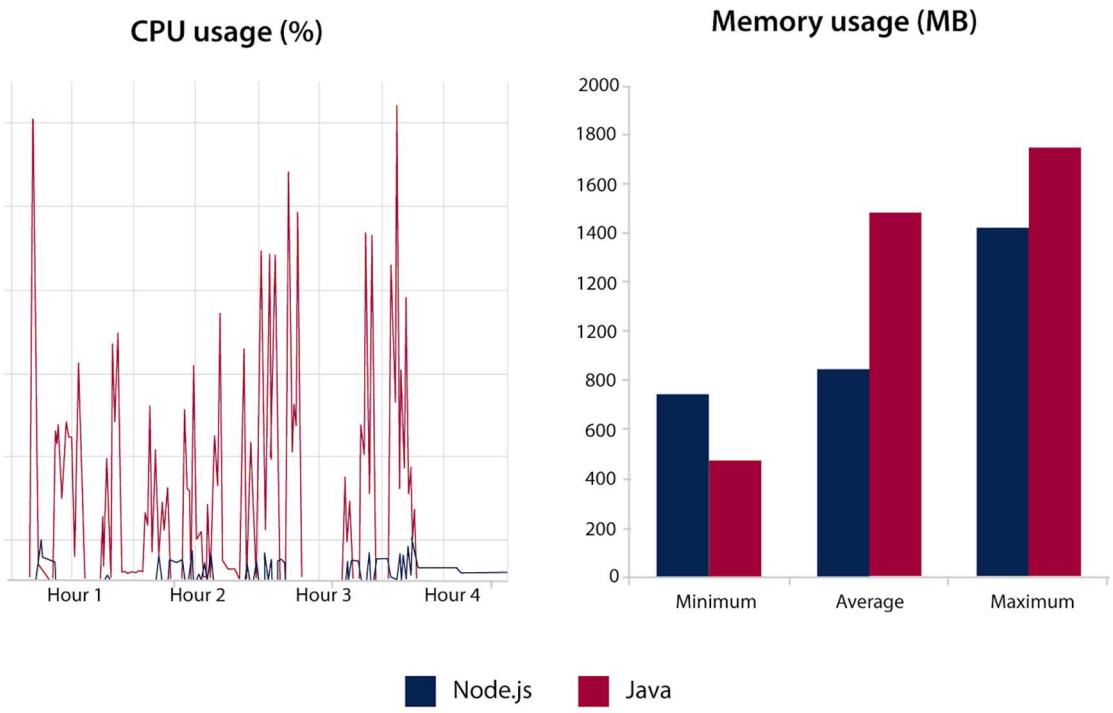


Рис. 2.2 Продуктивність Node.js та Java

Якщо запустити якусь просту задачу, на кшталт зведення в квадрат, то в тестах показники можуть відрізнятися до 10 разів. Якщо запустити цикли в мільйони завдань калькуляції, Java практично завжди буде перевершувати node.js. Величезна різниця між Java і node.js в тому, що node є однопоточним, це є як його перевагою, так і недоліком з іншого боку.

Java вміє працювати з потоками, які підтримуються на рівні ОС, і виходить, що програма написана на Java найбільш повно використовує можливості ОС. І якщо потрібно написати високонавантажених додаток, яке буде використовувати велику кількість обчислень, то Java для цього однозначно підійде краще. Проблема в тому, що навіть маленький сервер написаний на Java буде займати багато пам'яті - як на диску, так і оперативної.

Node.js є легковажним за рахунок архітектури побудованої на обробці подій. Він побудований для роботи в якості веб-сервера і дуже добре справляється з обслуговуванням легковажних завдань. Наприклад, простий запит на зразок розрахунку чого-небудь, або записи в базу даних відбувається дуже швидко. А якщо запитів стає дуже багато і ми хочемо масштабувати систему в node, можна використовувати веб-сервер Nginx або Apache. Можна завести багато однакових інстанси node. Тоді все буде розподілятися через балансування навантаження по round-robin. Якщо ми запустимо 8 інстанси node на 16 ядер відповідно, ОС сама розподілить інстанси між ядрами. Node цим не керує, у нього буде один потік.

Керування потоками в Java та Node.js. В Java ми можемо створити додаток і запустити в ньому 8 потоків. За рахунок того, що відбувається більш тісний контакт з ОС, можна розподілити навантаження. Як відомо, один з веб-серверів написаних на Java - це tomcat. Там можна чітко простежити, що коли користувач робить запит, запускаються додаткові потоки. А коли приходить запит на node, цикл подій (event loop) буде оброблений і відправлений назад, потім прийде наступний запит. І за рахунок того, що ми не чекаємо результатів першого, він теж буде підхоплений. Поки запити легкі, все добре. Однак, коли проводиться важке обчислення, при наявності одного інстанс, node зупиняється і настає тайм-аут. На рисунку 2.3 продемонстровано керування потоками в Java.

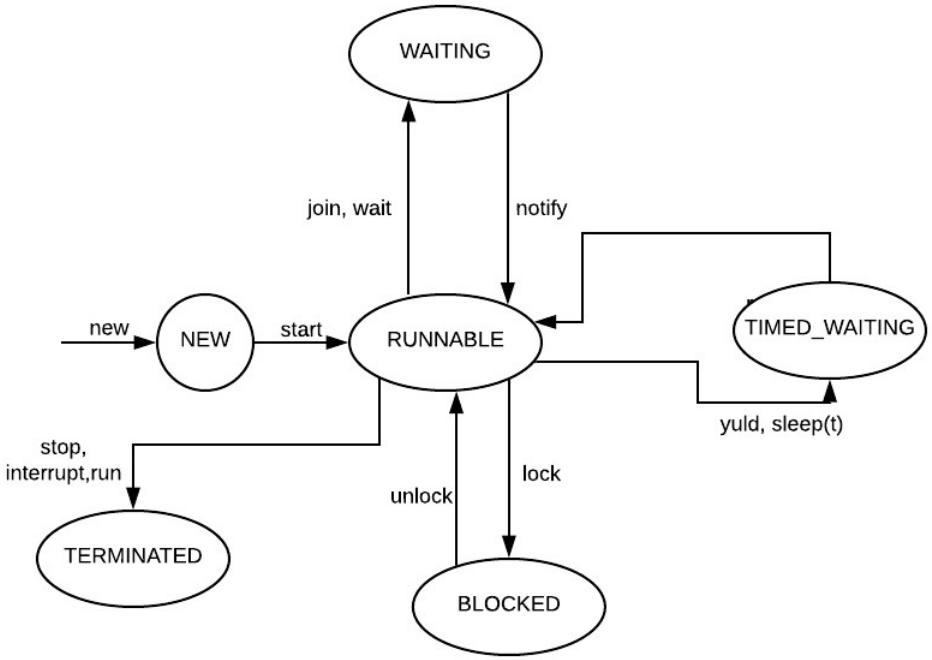


Рис. 2.3 Керування потоками в Java

На Node.js можна прописати буквально кілька рядків коду і отримати найпростіший веб-сервер. Природно, для більш широкого функціоналу, де будуть нотифікації, авторизації, тощо. це складніше реалізувати, але існують фреймворки які дозволяють вирішувати такі питання. На рисунку 2.4 продемонстровано керування потоками в Node.js.

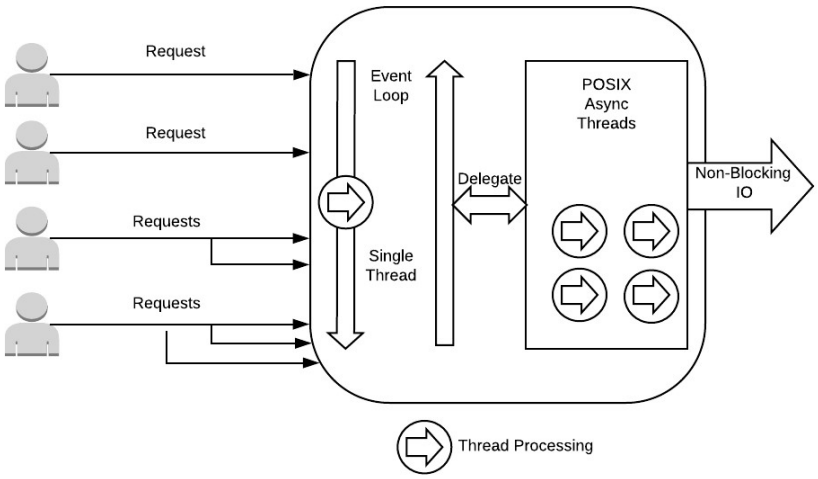


Рис 2.4 Керування потоками в Node.js

2.1.6. Express.js

Express.js - фреймворк web-додатків для Node.js, з відкритим кодом . Він спроектований для створення веб-додатків. Де-факто є стандартним каркасом для Node.js.

Засновником фреймворку є TJ Holowaychuk.Він описує цей фреймворк як створений на основі написаного на мові Ruby каркаса Sinatra, маючи на ув азі, що він мінімалістичний і включає велику кількість додаткових плагінів. Express може бути використаним для створення серверної частини для програмного стека MEAN, разом з базою даних MongoDB і каркасом Vue.js, React або AngularJS для клієнтської частини. Express.js надає ряд готових абстракцій, які спрощують створення сервера і серверної логіки, зокрема, обробка відправлених форм, робота з куками, CORS, тощо.

2.1.7. JWT

Веб-токен JSON або JWT є стандартом для безпечного передавання претензій у обмеженому простором середовищі. Він знайшов свій шлях до всіх основних веб-рамок. Простота, компактність та зручність використання - ключові особливості його архітектури. Хоча набагато складніші системи все ще використовуються, JWT мають широке коло застосувань.

Претензії - це визначення чи твердження, зроблені щодо певної сторони чи об’єкта. Деякі з цих тверджень та їх значення визначаються як частина специфікації JWT. Магія JWT полягає в тому, що вони стандартизують певні претензії, корисні в контексті деяких загальних операцій. Наприклад, одна з цих поширених операцій - встановлення особи певної сторони. Отже, одна із стандартних вимог, що зустрічаються в JWT, є предметом претензії.

Ще одним ключовим аспектом JWT є можливість їх підписання, використовуючи веб-підписи JSON (JWS, RFC 7515) та / або шифрувати їх, використовуючи веб-шифрування JSON (JWE, RFC 7516). Разом із JWS та

JWE, JWT забезпечують потужне, безпечне рішення для багатьох різних проблем.

Хоча головна мета JWT - передача претензій між двома сторонами, можливо, найважливішим аспектом цього є стандартизація зусиль у вигляді простого, необов'язково затвердженого та / або зашифрованого формату контейнера. Спеціальні рішення цієї ж проблеми в минулому впроваджувалися як приватно, так і публічно. Існують і більш давні стандарти встановлення претензій щодо певних сторін. Що JWT подає до таблиці, це простий, корисний, стандартний формат контейнера.

Веб-токен JSON виглядає приблизно так:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

JWT складається з трьох основних частин: заголовка (header), навантаження (payload) та підпис (signature). Тема і навантаження формуються окремо, а потім на їх основі обчислюється підпис.

Зазвичай заголовок складається з двох полів: типу токена (в даному випадку JWT) і алгоритму хешування підписи:

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

Офіційний сайт jwt.io рекомендує два алгоритму хешування: HS256 і RS256. Але на ділі можна використовувати будь-який алгоритм з приватним ключем.

Навантаження - це будь-які дані, які ви хочете передати в токені. Але стандарт передбачає кілька зарезервованих полів:

- iss - (issuer) - видавець токена

- sub - (subject) - «тема», призначення токена
- aud - (audience) - аудиторія, одержувачі токена
- exp - (expire time) - термін дії токена
- nbf - (not before) - термін, до якого токен не дійсний
- iat - (issued at) - час створення токена
- jti - (JWT id) - ідентифікатор токена

Всі ці поля не є обов'язковими, але їх використання не за призначенням може призвести до колізій.

Будь-які інші дані можна передавати по домовленості між сторонами, які використовують токен. Наприклад, навантаження може виглядати так:

```
{
  "iss": "Codex Team",
  "sub": "auth",
  "exp": 1505467756869,
  "iat": 1505467152069,
  "user": 1
}
```

Навантаження не шифрується при використанні токена, тому не варто передавати в ньому дані, які не повинні потрапити у відкритий доступ.

Підпис обчислюється на основі заголовка і навантаження. Таким чином, якщо хтось спробує змінити дані в токені, він не зможе змінити підпис, не знаючи приватного ключа. При аутентифікації приватним ключем може виступати пароль користувача (або хеш від пароля).

Спочатку заголовок і навантаження приводяться до формату JSON, а потім переводяться в base64:

```
Header: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
Payload:
eyJpc3MiOiJDb2RleCBUZWFtIiwic3ViIjoiYXV0aCIsImV4cCI6MTUwNTQ2Nzc1Njg2OSwiaWF0IjoxNTA1N
DY3MTUyMDY5LCJ1c2VyIjoxfQ
```

Потім, два ці рядки з'єднуються через точку і хешіруются зазначеним в header алгоритмом. Припустимо, користувач використовує пароль password:

```
HS256('eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9'+'.'+
'eyJpc3MiOiJDb2RleCBUZWFtIiwic3ViIjoiYXV0aCIsImV4cCI6MTUwNTQ2Nzc1Njg2OSwiaWF0IjoxNTA1
NDY3MTUyMDY5LCJ1c2VyIjoxfQ','password')='0ynjTRZT9Uk77TnGy_g9Mxi1decLBjKxQK6e2dVzDJo'
```

Результат роботи алгоритму і є підпис. Тепер залишилося тільки сформувати сам токен, для цього потрібно через точку з'єднати header і payload в base64 і підпис:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJDb2RleCBUZWFtIiwic3ViIjoiYXV0aCIsImV4
cCI6MTUwNTQ2Nzc1Njg2OSwiaWF0IjoxNTA1NDY3MTUyMDY5LCJ1c2VyIjoxfQ.0ynjTRZT9Uk77TnGy_g9Mx
i1decLBjKxQK6e2dVzDJo
```

Після першого логіна, клієнту повертається згенерований сервером JWT. При кожному наступному запиті, клієнт повинен передавати JWT встановленим API способом (наприклад, через заголовок або як параметр запиту). Сервер декодує заголовок і навантаження і перевіряє зарезервовані поля. Якщо все в порядку, за вказаним в заголовку алгоритмом складається підпис. Якщо отриманий підпис збігається з переданим, користувача авторизують.

2.2. Огляд Firebase

Firebase надає інструменти для розробки високоякісних додатків, розширення бази користувачів та заробітку. Cloud Firestore - це хмарна база даних NoSQL, до якої додатки iOS, Android та веб-програми можуть отримати доступ безпосередньо через рідні SDK. Cloud Firestore також доступний у рідних SDK-кодах Node.js, Java, Python, Unity, C ++ та Go, окрім REST API та RPC.

2.2.1. Технологія Cloud Firestore

Cloud Firestore - гнучка, масштабована хмарна база даних NoSQL для зберігання та синхронізації даних для розробки клієнтських та серверних служб.

Cloud Firestore - це гнучка, масштабована база даних для мобільних, веб- та серверних розробок від Firebase та Google Cloud Platform. Як і Firebase Realtime Database, вона підтримує синхронізацію між клієнтськими

програмами через слухачів у реальному часі та пропонує автономну підтримку для мобільних пристроїв та Інтернету, щоб розробник міг створювати чуйні додатки, які працюють незалежно від затримки мережі чи підключення до Інтернету. Cloud Firestore також пропонує безперешкодну інтеграцію з іншими продуктами Firebase та Google Cloud Platform, включаючи хмарні функції.

Можливості Cloud Firestore:

- Гнучкість. Модель даних Cloud Firestore підтримує гнучкі, ієрархічні структури даних. Дані зберігаються в документах, впорядкованих у колекції. Документи можуть містити складні вкладені об'єкти на додаток до підколекцій.
- Виразний запит. У Cloud Firestore ви можете використовувати запити для отримання окремих, конкретних документів або для отримання всіх документів у колекції, що відповідають параметрам запиту. Запити можуть включати в себе декілька ланцюгових фільтрів та комбінувати фільтрування та сортування. Вони також індексуються за замовчуванням, тому ефективність запитів пропорційна розміру вашого набору результатів, а не набору даних.
- Оновлення у режимі реального часу. Як і база даних в режимі реального часу, Cloud Firestore використовує синхронізацію даних для оновлення даних на будь-якому підключеному пристрої.
- Офлайн-підтримка. Cloud Firestore кешує дані, які активно використовує додаток, тому програма може записувати, читати, слухати та запитувати дані, навіть якщо пристрій перебуває в режимі офлайн. Коли пристрій повернеться в Інтернет, Cloud Firestore синхронізує всі локальні зміни назад у Cloud Firestore.
- Розроблена для масштабування. Cloud Firestore містить найкраще з потужної інфраструктури платформи Google Cloud: автоматична реплікація даних у багатьох регіонах, чіткі гарантії послідовності, атомні пакетні операції та реальна підтримка транзакцій.

2.2.2. Технологія Cloud Functions

Cloud Functions - це безсерверний фреймворк, який дозволяє автоматично запускати резервний код у відповідь на події, викликані функціями Firebase та HTTPS-запитами. Код JavaScript або TypeScript зберігається в хмарі Google і працює в керованому середовищі. Це дозволяє мінімізувати керування та масштабування власних серверів.

Можливості Cloud Functions:

- Інтегрує платформу Firebase. Функції можуть реагувати на події, породжені різними функціями Firebase та Google Cloud, від тригерів аутентифікації Firebase до тригерів Cloud Storage. Хмарні функції мінімізують код, полегшуючи використання Firebase та Google Cloud у межах користувацької функції.
- Мінімальне обслуговування. JavaScript або TypeScript-код можливо розгорнути на серверах за допомогою однієї команди з командного рядка. Після цього Firebase автоматично масштабує обчислювальні ресурси, щоб відповідати моделям використання користувачів. Не має потреби не турбуватись про облікові дані, конфігурацію сервера, надання нових серверів або виведення з експлуатації старих.
- Зберігає логіку приватною і захищеною. У багатьох випадках розробники вважають, що краще керувати логікою програми на сервері, аби уникнути підробки на стороні клієнта. Крім того, іноді не бажано дозволяти індексувати цей код. Cloud Functions повністю захищені від клієнта, тож розробник може бути впевненим, що вони приватні і завжди роблять саме те, що він хоче.

Коли навантаження збільшується чи зменшується, Google реагує, швидко зменшуючи кількість екземплярів віртуального сервера, необхідних для запуску користувацької функції. Кожна функція працює ізольовано, у власному середовищі зі своєю власною конфігурацією.

Життєвий цикл фонової функції:

1. Розробник пише код для нової функції, вибираючи постачальника подій (наприклад, база даних в реальному часі) та визначаючи умови, за яких функція повинна виконуватись.
2. Розробник розгортає функцію, а Firebase підключає її до обраного постачальника подій.
3. Коли постачальник подій генерує подію, яка відповідає умовам функції, викликається код.
4. Якщо функція зайнята керуванням багатьма подіями, Google створює більше примірників, щоб швидше працювати з функцією. Якщо функція не працює, екземпляри очищуються.
5. Коли розробник оновлює функцію, розгортаючи оновлений код, всі екземпляри старої версії очищуються та замінюються новими екземплярами.
6. Коли розробник видаляє функцію, всі екземпляри очищуються, а з'єднання між функцією та провайдером подій видаляється.

Крім прослуховування подій з фоновою функцією, розробник може викликати функцію безпосередньо за допомогою HTTP-запиту або виклику від клієнта.

2.2.3. Технологія Firebase Authentication

Більшість додатків повинні знати особу користувача. Знання ідентичності користувача дозволяє додатку надійно зберігати дані користувачів у хмарі та надавати однаковий персоналізований досвід на всіх пристроях користувача.

Перевірка автентичності Firebase надає сервіси, прості у використанні SDK та готові бібліотеки інтерфейсу для аутентифікації користувачів вашої програми. Він підтримує аутентифікацію, використовуючи паролі, номери

телефонів, популярних федеральних постачальників ідентифікаційних даних, таких як Google, Facebook та Twitter тощо.

Аутентифікація Firebase тісно інтегрується з іншими службами Firebase і використовує такі галузеві стандарти, як OAuth 2.0 та OpenID Connect, тому вона може бути легко інтегрована у власний сервер.

Можливості Cloud Authentication:

- FirebaseUI надає розв'язувальне рішення для автентичності, яке обробляє потоки інтерфейсу користувача для входу користувачів з адресами електронної пошти та паролями, номерами телефонів, а також з популярними федеральними постачальниками ідентифікаційних даних, включаючи вхід в систему Google та Facebook.
- Компонент FirebaseUI Auth реалізує кращі практики аутентифікації на мобільних пристроях та веб-сайтах, що дозволяє максимально ввійти та конвертувати реєстрацію для програми. Він також обробляє виключні ситуації, такі як відновлення облікового запису та з'єднання облікових записів, які можуть бути чутливими до безпеки та схильні до помилок.
- FirebaseUI можна легко налаштувати так, щоб він відповідав решті візуального стилю додатка.
- Автентифікація на основі електронної пошти та пароля. Пакет SDK для аутентифікації Firebase надає методи створення та керування користувачами, які використовують для входу їх адреси електронної пошти та паролі. Ідентифікація Firebase Authentication також обробляє надсилання електронних листів для скидання пароля.
- Інтеграція постачальника об'єднаних ідентифікаторів. Cloud Authenticaton надає можливість аутентифікації користувачів, інтегруючись із об'єднаними постачальниками ідентичності. SDK

Firebase Authentication пропонує методи, які дозволяють користувачам входити через свої облікові записи Google, Facebook, Twitter та GitHub.

- Аутентифікація телефонного номера. Можливість аутентифікувати користувачів, надсилаючи SMS-повідомлення на їх телефони.
- Анонімна авторизація. Можливість використання функції, для яких потрібна аутентифікація, щоб користувачі спочатку входили, створюючи тимчасові анонімні облікові записи. Якщо пізніше користувач вирішить зареєструватися, розробник може оновити анонімний обліковий запис до звичайного облікового запису, щоб користувач міг продовжувати там, де зупинився.

Щоб увійти в додаток користувача, спочатку отримуються облікові дані аутентифікації від користувача. Ці облікові дані можуть бути електронною адресою користувача та паролем користувача або маркером OAuth від федерального постачальника ідентифікаційних даних. Потім ви передаєте ці дані в SDK для аутентифікації Firebase. Потім сервіси перевірять ці дані та повернуть відповідь клієнту.

Після успішного входу користувач може отримати доступ до основної інформації, а також може контролювати доступ до даних, що зберігаються в інших продуктах Firebase. Також можна використовувати наданий маркер автентифікації для перевірки особи користувачів у власних сервісах.

2.2.4. Технологія Firebase Hosting

Firebase Hosting забезпечує швидкий і безпечний хостинг для веб-додатку, статичного та динамічного вмісту та мікросервісів.

Firebase Hosting - це хостинг веб-контенту для розробників. За допомогою однієї команди можна швидко розгортати веб-програми та обслуговувати як статичний, так і динамічний контент у глобальній CDN (мережі доставки вмісту). Також можна поєднати хостинг Firebase з Cloud

Functions або Cloud Run для створення та розміщення мікросервісів на Firebase.

Можливості Firebase Hosting:

- Передача контенту через захищене з'єднання. Нульова конфігурація SSL вбудована в хостинг Firebase, тому контент завжди надійно захищений.
- Статичний та динамічний контент. Сервіси Firebase Hosting підтримують всі типи контенту для хостингу, від CSS та HTML-файлів до мікросервісів Express.js або API.
- Швидка доставка вмісту. Кожен завантажений файл зберігається на краях CDN по всьому світу. Незалежно від місця користувачів, вміст доставляється швидко.
- Розгортка нових версій за допомогою однієї команди. За допомогою CLI Firebase розробник може запустити додаток за лічені секунди. Інструменти командного рядка полегшують додавання цілей розгортання у процес збирання.
- Відкат у один клік. Швидкі розгортання чудові, але можливість скасувати помилки ще краще. Firebase хостинг забезпечує повну версію версій та управління випуском за допомогою зворотних зворотів одним кліком.

Firebase хостинг створений для сучасного веб-розробника. Веб-сайти та додатки є більш потужними завдяки піднесенню фреймворків JavaScript, таких як Angular та статичних генераторних інструментів, як Jekyll. Незалежно від того, чи розгортає розробник просту цільову сторінку додатка або складний прогресивний веб-додаток (PWA), хостинг надає інфраструктуру, функції та інструменти, призначені для розгортання та керування веб-сайтами та програмами.

Firebase хостинг має легкі варіанти конфігурації хостингу для створення складних PWA. Розробник може легко переписати URL-адреси для маршрутизації на стороні клієнта або налаштувати власні заголовки.

2.3 Бази даних

База даних - це формований збір структурованої інформації або даних. Зазвичай запам'ятовується в електронному вигляді в комп'ютерній системі. База даних зазвичай контролюється системою управління базами даних (СУБД). Разом інформація та СУБД, разом із програмами, які пов'язані з ними, називаються системою баз даних, часто скорочується до просто бази даних.

Дані в найпоширеніших типах баз даних, що діють сьогодні, зазвичай моделюються у рядках та стовпцях у ряді таблиць, щоб зробити о бробку та запити даних ефективними. Далі можна легко отримати доступ, керувати, змінювати, оновлювати, контролювати та організовувати дані. Більшість баз даних використовують структуровану мову запитів (SQL) для запису та запиту даних.Комп'ютерні програми можуть класифікувати систему управління базами даних у відповідності з моделями базових дан их, які вони підтримують.

Реляційні бази даних стали домініруючими в 1980х роках. Ці дані моделюються у вигляді малюнка та стільниці у ряді таблиць, і надаючи велике використання використовує SQL для записів та запитуючих даних. У 2000х роках були популярні нереляційні бази даних, які називаються NoSQL, за їх допомогою вони використовують різні мови.

2.3.1 Що таке структурована мова запитів (SQL)?

SQL - це мова програмування, що використовується майже у всіх реляційних базах даних для запиту, обробки і визначення даних, а також для забезпечення контролю доступу. SQL був вперше розроблений в IBM в 1970х роках, коли Oracle виступив в якості основного учасника, що призвело до

впровадження стандарту SQL ANSI. SQL стимулював багато розширення таких компаній, як IBM, Oracle і Microsoft. Хоча SQL все ще широко використовується сьогодні, нові мови програмування починають з'являтися.

2.3.2 Еволюція баз даних

Бази даних значно змінилися з моменту їх створення на початку 1960х років. Навігаційні бази даних, такі як ієрархічна база даних (яка спиралася на деревоподібну модель і допускала тільки ставлення «один до багатьох») і мережева база даних (більш гнучка модель, яка припускає множинні відносини), були вихідними системами, використовуваними для зберігання і маніпулювати даними. Незважаючи на простоту, ці ранні системи були негнучкими. У 1980х роках реляційні бази даних стали популярними, за ними послідували об'єктно-орієнтовані бази даних в 1990х роках.

Зовсім недавно бази даних NoSQL стали відповіддю на зростання Інтер нету і необхідність в більшої швидкості і обробці неструктурованих даних. С ьогодні хмарні бази даних і автономні бази даних відкривають нові можливос ті, коли мова йде про збір, зберігання, управлінні і використанні даних.

2.3.3 Яка різниця між базою даних та електронною таблицею?

Бази даних та електронні таблиці (такі як Microsoft Excel) є зручними способами зберігання інформації.

Основні відмінності між ними:

- Як дані зберігаються і управляються
- Хто може отримати доступ до даних
- Скільки даних можна зберігати

Таблиці спочатку були призначені для одного користувача, і їх характеристики відображають це. Вони відмінно підходять для одного користувача або невеликого числа користувачів, яким не потрібно виконувати неймовірно складні маніпуляції з даними. Бази даних, з іншого боку,

призначені для зберігання набагато більших колекцій організованою інформації - іноді величезних обсягів. Бази даних дозволяють кільком користувачам одночасно швидко і безпечно отримувати доступ і запитувати дані, використовуючи дуже складну логіку і мову.

2.3.4 Типи баз даних

Існує багато різних типів баз даних. Найкраща база даних для конкре тної організації залежить від того, як організація має намір використовувати дані.

- Реляційні бази даних. Реляційні бази даних стали домінуючими в 1980х роках. Елементи в реляційній базі даних організовані у вигляді набору таблиць із стовпцями та рядками. Технологія реляційних баз даних забезпечує найбільш ефективний та гнучкий спосіб доступу до структурованої інформації.
- Об'єктно-орієнтовані бази даних. Інформація в об'єктно-орієнтованій базі даних представлена у вигляді об'єктів, як у об'єктно-орієнтованому програмуванні.
- Розподілені бази даних. Розподілена база даних складається з двох або більше файлів, розташованих на різних сайтах. База даних може зберігатися на декількох комп'ютерах, розташованих в одному фізичному місці, або розкидатися по різних мережах.
- Склади даних. Центральний сховище даних, сховище даних - це тип бази даних, спеціально розроблений для швидкого запиту та аналізу.
- Бази даних NoSQL. NoSQL або нереляційна база даних дозволяє зберігати і маніпулювати неструктурованими і напівструктурованими даними (на відміну від реляційної бази даних, яка визначає, як повинні складатися всі дані, вставлені в базу даних). Бази даних NoSQL стали популярнішими, оскільки веб-додатки ставали все більш поширеними та складнішими.

- Графічні бази даних. Графічна база даних зберігає дані за сутностями та відносинами між сутностями.
- OLTP бази даних. База даних OLTP - це швидка аналітична база даних, розроблена для великої кількості транзакцій, здійснюваних кількома користувачами.

Це лише декілька з кількох десятків типів баз даних, які використовуються сьогодні. Інші, менш поширені бази даних адаптовані до дуже конкретних наукових, фінансових чи інших функцій. Окрім різних типів баз даних, зміни в підходах до розробки технологій та значні досягнення, такі як хмара та автоматизація, приводять в рух нові бази даних в абсолютно нових напрямках. Деякі з останніх баз даних включають:

- Бази даних з відкритим кодом. Система баз даних з відкритим кодом - це система, вихідний код якої є відкритим кодом; такі бази даних можуть бути базами даних SQL або NoSQL.
- Хмарні бази даних. Хмарна база даних - це сукупність даних, структурованих або неструктурованих, що розміщується на приватній, загальнодоступній або гібридній хмарній платформі хмарних обчислень. Існує два типи хмарних моделей баз даних: традиційна та база даних як послуга (DBaaS). За допомогою DBaaS адміністративні завдання та обслуговування виконує постачальник послуг.
- Мульти-модель бази даних. Мульти-модельні бази даних поєднують різні типи моделей баз даних в єдиний, інтегрований задній край. Це означає, що вони можуть вміщувати різні типи даних.
- Документ / база даних JSON Бази даних документів, призначені для зберігання, пошуку та управління інформацією, орієнтованою на документ - це сучасний спосіб зберігання даних у форматі JSON,

а не рядків та стовпців.

- Самостійні керування базами даних. Найновіший і найбільш новаторський тип баз даних, самостійно керовані бази даних (також відомі як автономні бази даних) є хмарними та використовують машинне навчання для автоматизації налаштування баз даних, захисту, резервного копіювання, оновлень та інших рутинних завдань управління, які традиційно виконуються адміністраторами баз даних.

2.3.5 Що таке система управління базами даних?

База даних зазвичай вимагає комплексного програмного забезпечення бази даних, відомого як система управління базою даних (СЕБД). СУБД служить інтерфейсом між базою даних і її кінцевими користувачами або програмами, дозволяючи користувачам отримувати, оновлювати і управляти тим, як інформація організована і оптимізована. СУБД також полегшує контроль і управління базами даних, дозволяючи виконувати різні адміністративні операції, такі як моніторинг продуктивності, настройка, а також резервне копіювання і відновлення.

Деякі приклади популярних програм баз даних або СУБД включають MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, Oracle Dat abase і dBASE.

2.3.5 Що таке база даних MySQL?

MySQL - це система керування базами даних з відкритим вихідним кодом, заснована на SQL. Він був розроблений і оптимізованийдля веб-додатків і може працювати на будь-якій платформі. У міру появи нових і різних вимог в Інтернеті MySQL стала платформою вибору для веб-розробників і веб-додатків. Оскільки MySQL розроблений для обробки мільйонів запитів і тисяч транзакцій, він є популярним вибором для підприємств електронної комерції, яким необхідно управляти декількома грошовими переказами. Гнучкість на вимогу є основною функцією MySQL.

MySQL - це СУБД, що стоїть за деякими провідними веб-сайтами та веб-додатками в світі, включаючи Airbnb, Uber, LinkedIn, Facebook, Twitter і YouTube.

2.3.6 Виклики бази даних

Сучасні корпоративні бази даних часто підтримують дуже складні запити і, як очікується, будуть надавати майже миттєві відповіді на ці запити. В результаті до адміністраторів баз даних постійно звертаються з проханням використовувати найрізноманітніші методи для підвищення продуктивності.

Деякі загальні проблеми, з якими вони стикаються, включають в себе:

- Поглинання значного збільшення обсягу даних. Вибух даних, що надходять від датчиків, підключених комп'ютерів і десятків інших джерел, змушує адміністраторів баз даних ефективно керувати даними своїх компаній і організовувати їх.
- Забезпечення безпеки даних. В даний час повсюдне поширення даних відбувається, і хакери стають все більш винахідливими. Як ніколи важливо забезпечити безпеку даних, а також їх легкий доступ для користувачів.
- Йти в ногу з попитом. У сучасній швидко мінливого бізнес-середовищі компаніям необхідний доступ до своїх даних в режимі реального часу для підтримки своєчасного прийняття рішень і використання нових можливостей.
- Управління та підтримка бази даних та інфраструктури. Адміністратори бази даних повинні постійно відслідковувати наявність проблем в базі даних і виконувати профілактичне обслуговування, а також встановлювати оновлення та виправлення програмного забезпечення. Оскільки бази даних стають все більш складними і обсяги даних ростуть, компанії стикаються з необхідністю залучення додаткових фахівців для моніторингу і налаштування своїх баз даних.

- Зняття обмежень на масштабованість. Бізнес повинен розвиватися, якщо він збирається вижити, і його управління даними повинно зростати разом з ним. Але адміністраторам баз даних дуже складно передбачити, скільки потужності потрібно компанії, особливо за допомогою локальних баз даних.

Рішення всіх цих проблем може зайняти багато часу і може перешкодити адміністраторам баз даних виконувати більш стратегічні функції.

ВИСНОВОК ДО РОЗДІЛУ 2

У розділі 2 було проведено опис предметної області проекту.

Було проаналізовано бібліотеку Node.js та движок V8, на якому побудована бібліотека. Також була проаналізована різниця між Node.js та Java, їх недоліки та переваги. Ще був проаналізований фреймворк Express.js, технологія Websocket, а ще технологія JSON Web Token(JWT).

Також був проведений аналіз технологій Firebase, а саме:

- Cloud Firestore
- Cloud Functions
- Firebase Authentication
- Firebase Hosting

У цьому розділі також розібрано що таке бази даних, що таке мова запитів SQL, які бувають типи баз даних, еволюцію баз даних, що таке СУБД і популярну платформу MySQL.

РОЗДІЛ 3.
РОЗРОБКА ДОДАТКУ

3.1. Вибір технологій та їх обґрунтування

3.1.1. Вибір платформи для додатку

У розділах 1 та 2 були визначені вимоги щодо проектування серверної частини системи безпеки розумного дому з використанням web-socket та були поставлені такі завдання:

1) Додаток повинен бути легким та простим у використанні.
2) Додаток повинен підтримуватися більшістю сучасних браузерів та мобільних пристроїв.
3) Додаток повинен працювати швидко.
4) Додаток повинен приваблювати людей, які користуються IoT-пристроями.

Для того, щоб усі вимоги були дотримані, необхідно обрати серверну платформу для розробки. Вона повинна бути надійна та швидка. Серед запропонованих мов програмування для серверних систем є такі популярні мови: Python, Java та Node.js.

Розглянемо кожну з перелічених мов, для того, щоб обрати таку, на якій розробка серверної частини системи безпеки розумного дому буде максимально комфортною та легкою для розробника.

Python виник на початку 90-х років, до цих пір залишається однією з найбільш інноваційних, гнучких і універсальних технологій завдяки бібліотекам, чудовою документацією і новітньою реалізацією. Python є основною мовою для науки про дані, машинного навчання та проектів штучного інтелекту.

Python також володіє однією з найбільших спільнот, яка працює над поліпшенням мови для вирішення сучасних завдань програмування.

Переваги Python:

- Python скорочує час виходу на ринок. За допомогою Python можна розробляти MVP або прототип в обмежені терміни для скорочення часу виходу на ринок. Цього можна досягти завдяки методу швидкої розробки Python, який дозволяє підтримувати декілька ітерацій одночасно, і принципу DRY, що передбачає можливість повторного використання частин коду.
- Простота синтаксису Python. Одна з головних причин любові розробників до Python полягає в простоті синтаксису, за допомогою якого можна висловити концепції в декількох рядках коду, що полегшує вирішення помилок і налагодження. Python - це про читабельність коду. Він також досить простий для розуміння клієнтами, що полегшує співпрацю.
- Широкий спектр інструментів розробки і фреймворків Python. Популярний редактор коду Sublime Text забезпечує підтримку розробки на Python, а також додаткові функції редагування і розширення синтаксису. Потужні веб-фреймворки спрощують процес і дозволяють розробникам зосередитися на логіці додатків.
- Велике співтовариство. У порівнянні з Node.js, Python є більш зрілим open-source мовою і володіє одним з найбільших спільнот з неймовірною кількістю учасників: від новачків до досвідчених фахівців, які діляться рішеннями і покращують мову.

Недоліки Python:

- Python є однопоточною мовою. Як і будь-яка інтерпретована мова, Python має більш повільну швидкістю виконання в порівнянні з компільованими мовами (такими як C або Swift). Він може бути не

кращим вибором для додатків з безліччю складних обчислень або для будь-якого проекту, де швидкість виконання є найбільш важливою вимогою.

- Слабкість в мобільних обчисленнях. Python відмінно підходить для розробки серверних і настільних платформ, але він вважається слабким в мобільних обчисленнях.

Python підходить для всіх видів проектів: маленьких і великих, простих і складних, включаючи бізнес-додатки, настільні інтерфейси, освітні платформи, ігрові та наукові програми.

Node.js - це середовище, яке дозволяє використовувати JavaScript як для бекенд, так і для фронтенд розробки, а також для вирішення проблем сумісності. Node.js також можна визначити як мову сценаріїв на стороні сервера. Він був запущений в 2009 році і поступово набирає популярність.

Переваги Node.js:

- Node.js забезпечує високу продуктивність. Відносно швидкості Node.js швидше, ніж Python. Він заснований на движку Google V8, що робить його придатним для розробки чат-ботів і аналогічних програм реального часу.
- Надає можливість full-stack розробки. Для створення програми потрібна лише одна команда розробників, які знають JavaScript, що дозволяє скоротити витрати.
- Дуже добре підходить для розробки додатків реального часу. Завдяки подієво-орієнтованій архітектурі можна розробляти чат-додатки і веб-ігри.

Недоліки Node.js:

- Node.js вимагає чистої архітектури. Це подієво-орієнтоване середовище, тому воно може запускати кілька подій одночасно, але тільки якщо відносини між ними добре прописані.
- Не підтримує завдання з високим завантаженням процесора. Важкий обчислювальний запит блокує обробку інших завдань і уповільнює роботу програми, написаної на Node. Тому він не підходить для проектів, заснованих на науці про дані.
- Недостатньо розвинена документація. На відміну від Python, документація Node.js розвинена недостатньо. До того ж Node.js не володіє основними бібліотеками та інструментами, а через велику кількість альтернатив не завжди зрозуміло, який варіант варто обрати.

Node.js - це ключова технологія для розробки додатків, таких як рекламні сервіси, ігрові платформи або форуми. Він підходить для обробки проектів з безліччю одночасних підключень або додатків з високошвидкісним введенням/виведенням, а також таких додатків, як платформи для підвищення продуктивності (наприклад, системи управління контентом), торгові майданчики P2P і платформи електронної комерції.

Використання Java на стороні сервера відкриває ще одну перевагу даної технології. При необхідності внесення змін до бізнес-логіки, серверні додатки Java, розроблені на основі компонентів Enterprise JavaBeans (EJB), дозволяють істотно полегшити процедуру оновлення прикладного ПО з появою нових версій. Певні переваги від використання Java можуть отримувати і вже традиційні обчислювальні технології, такі, як СУБД, де Java все частіше використовується як процедурна мова на сервері БД.

Як відомо, Java може використовуватися на обох полюсах клієнт-серверної технології: на робочому місці у користувача, наприклад в Web-

браузері, і на сервері в мережі. Незважаючи на те, що застосування Java на стороні клієнта стало звичайною справою, широке поширення цієї мови все ще стримується недостатньою його безпекою та високими вимогами до смуги пропускання. Комбінація Java-додатків на стороні сервера і динамічної генерації HTML-потоку усталилася як найбільш практична технологічна реалізація бізнес-логіки, що забезпечує достатню безпеку.

Обираючи технології Java для серверної сторони доводиться враховувати вже набагато більшу кількість чинників. Необхідно обрати найбільш придатний Web-сервер і монітор транзакцій. У ідеалі модель серверних додатків Java не обмежується можливостями якогось певного середовища розробки. При виборі серверної інфраструктури Java вам у першу чергу необхідно визначитися з тим колом завдань, які вирішуватиме створювана вами обчислювальне середовище Java.

Серйозною перевагою використання технології Java на сервері є те, що вона надає уніфіковані API і середу виконання, не прив'язані до якогось одного постачальника, і в той же час дозволяє уникнути зниження продуктивності, характерного для інтерфейсу CGI при інтенсивному зверненні через нього до серверних додатків (кожен додаток Java запускається в якості полегшеного потоку). Серверні Java-додатки виконуються у власному віртуальному середовищі, яке забезпечує безпеку і можливість роботи з прикладними службами. І оскільки віртуальна Java-машина виконується як окремий користувальницький процес, помилки в окремих серверних додатках, створених на базі інтерфейсів NSAPI / ISAPI (Netscape API / Internet Server API), не зможуть привести до руйнування або зупинки основного процесу Web-сервера. Крім того, в розпорядженні розробника, який обрав цю технологію, виявляється багатий набір класів, інструментів розробки і програмних інтерфейсів.

Модель побудови інформаційної системи на базі серверних додатків Java має свої особливості. Одна з них - обмежена продуктивність технології.

Однак, працюючи на сервері додатків, який одночасно є і Web-сервером, компоненти Java дозволяють виконувати контекстно-залежне перемикання між Web-сервером і Java-машиною. Крім того, додатки, що використовують програмні інтерфейси NSAPI / ISAPI, можуть бути скомпільовані безпосередньо в виконуваний код, що забезпечує набагато більшу швидкодію програм, ніж інтерпретація байт-коду.

Механізм серверного додатка - це JVM, що забезпечує роботу Java-програм через API і протокол зв'язку з процесом Web-сервера. API для серверних додатків складається з двох пакетів: javax.servlet і javax.servlet.http. Віртуальна Java-машина для сервера додатків (вона зазвичай запускається на тому ж Web-сервері) відповідає за завантаження ваших класів, а також за управління багатопоточністю і клієнтськими запитами. Серверна JVM створює екземпляри об'єктів, виконує їх налаштування відповідно до конфігурацій і організовує сеанси зв'язку. Вона ж вивантажує відпрацьовані об'єкти і здійснює операцію "збірки сміття". Програмний інтерфейс для серверних додатків Java містить прості об'єкти, що забезпечують видачу клієнтам HTML-файлів, відновлення запитів клієнтів і масу інших допоміжних функцій: запит персональної інформації у клієнта, відстеження сеансів, управління багатопоточністю і ін.

Стандартна Java-машина для серверних додатків зазвичай реалізує базисні функції, такі, наприклад, як підтримка сеансів зв'язку і запит персональної інформації, в той час як стандартні Java-класи, зокрема JDBC (Java Database Connectivity) - доступ до сховищ корпоративних даних. ПО WebSphere, втім, як і GemStone / J, розширює базові можливості сервера Java-додатків, надаючи кошти управління і конфігурації, базовий набір обробки транзакцій, а також службу кешування об'єктів.

Проаналізувавши переваги платформ для розробки серверної частини системи безпеки розумного дому як Java, Python та Node.js, було зроблено

висновок, що для розробки серверної частини системи краще використовувати Node.js.

Такий вибір має декілька переваг:

1) Можливість застосовувати одну мову на клієнті і сервері.
2) Швидкість.
3) Синтаксис Javascript.
4) Асинхронність.
5) Велика масштабованість.

3.1.2. Вибір мови програмування

Для розробки серверної частини системи безпеки розумного дому було обрано таку мову програмування Javascript і бібліотеку Node.js, адже це одна з найбільш популярних і потужних бібліотек, використовуваних для розробки серверів.

3.1.3. Вибір допоміжних бібліотек

1) Node.js – бібліотека, яка допомагає Javascript працювати з пристроями вводу-виводу. У даній роботі він потрібен для будування веб-сервера;
2) Express.js – фреймворк веб-додатків. Він потрібен для використання системи маршрутизації для веб-сервера;
3) Firebase-admin – бібліотека для використання Firestore;
4) Express-ws – бібліотека для використання технології websocket з фреймворком Express.js;
5) Body-parser – бібліотека для обробки тіла POST-запиту;
6) jsonwebtoken – бібліотека яка надає функції для роботи з JSONWebToken(JWT), надалі буде jwt;

3.2. Основні рішення з реалізації додатку та його компонентів

Розробку додатку можна поділити на такі пункти:

1. Реалізація API для сторінки входу;
2. Реалізація API для сторінки реєстрації;
3. Реалізація API для головної сторінки додатку;
4. Реалізація API для сторінки всіх пристроїв;
5. Реалізація API для сторінки кімнат.

3.2.1 Реалізація функцій для використання JSONWebToken

На рисунку 3.1 продемонстровано код для перевірки валідності токена.

```
const jwt = require('jsonwebtoken');

const isTokenExpired = (res, exp) => {
    new Promise((resolve, reject) => {
        var current_time = new Date().getTime() / 1000;
        if (current_time > exp) {
            throw 401;
        } else {
            console.log('not exp');
        }
    })
    .catch(status => res.sendStatus(status));
}
```

Рис. 3.1 Функція перевірки валідності токена

Функція isTokenExpired перевіряє, якщо час дії токена сплинув, то вона відправляє на клієнт статус 401, що означає що авторизація не пройдена. В іншому випадку воно виводить на у консоль на сервері повідомлення, що час дії токена ще не сплинув. Змінна current_time це змінна для поточного часу. Аргумент exp приймає змінну exp з розшифрованого корисного навантаження токена, який прийшов з клієнту. Ця змінна несе в собі данні через який час токен стане не дійсним.

На рисунку 3.2 показана функція верифікації токена.

```
const tokenVerification = (token, res) => {
    jwt.verify(token, 'secret', function(err, decode) {
        if(err) {
            res.set(401).send(err);
        } else {
            isTokenExpired(res, decode.exp)
        }
    })
}
```

Рис. 3.2 Функція верифікації токена

Верифікація токена потрібна для підтвердження валідності токена. Функція з бібліотеки jsonwebtoken jwt.verify приймає три аргументу:

1. Перший аргумент це сам токен.
2. Другий аргумент це секретний ключ. Він потрібен для того, будь-яка людина не змогла розшифрувати токен і дані, які він в собі несе. Токен можливо розшифрувати лише за допомогою цього секретного ключа.
3. Третій аргумент приймає колбек функцію, яка викликається з декодованим корисним навантаженням, якщо підпис дійсний і допустимі закінчення строку дії. Якщо ні - він буде викликаний з помилкою.

Якщо верифікація виконується з помилкою то на клієнт відправляється відповідь зі статусом 401 – неавторизований, і дані помилки. Якщо ж ні, то виконується функція, описана вище.

3.2.2 Реалізація функцій, для використання Firestore

```
let db = admin.firestore();

const getData = collectionName => {
    return db.collection(collectionName).get()
        .then(snapshot => {
            const data = []
            snapshot.forEach(doc => data.push(doc.data()))
            return data;
        })
        .catch((err) => {
            console.log('Error getting documents', err);
        });
}
```

Рис. 3.3 Функція отримання даних з бази даних

Функція getData приймає один аргумент collectionName, який повинен нести в собі рядок назви потрібної колекції з якої потрібно отримати дані. Функція db.collection(collectionName).get() повертає Promise, у якому можливо обробляти масив документів колекції(snapshot). Далі функція повертає данні з колекції, які далі можливо використовувати для подальших обчислювань.

```
const addData = (collectionName, docID, data) => {
    db.collection(collectionName).doc(docID).set(data)
}
```

Рис 3.4 Функція додавання даних до бази даних

На рисунку 3.4 продемонстровано функція додавання даних до бази даних. Функція addData приймає три аргументи:

1. collectionName – ім’я колекції, у яку потрібно додати дані;
2. docID – ID документа, який треба додати, це може буди як і рядок, так і число.
3. data – дані, які потрібно додати.

```
const changeStatus = (docID) => {
    const ref = db.collection('users-devices').doc(docID);
    return db.runTransaction(t => {
        return t.get(ref)
            .then(doc => {
                console.log(doc.data().status);
                switch (doc.data().status) {
                    case 'active':
                        t.update(ref, {status: 'disabled'});
                        return Promise.resolve('Device status disabled');
                    case 'disabled':
                        t.update(ref, {status: 'active'});
                        return Promise.resolve('Device status activeч');
                }
            })
    })
    .then(result => console.log('Transaction success!', result))
    .catch(err => console.log('Transaction failure:', err))
}
```

Рис. 3.5 Функція зміни статусу певного девайсу.

На рисунку 3.5 продемонстрована функція зміни статусу певного девайсу. Функція changeStatus приймає один аргумент – docID. Він потрібен для того щоб знайти певний девайс в базі даних та змінити в ньому статус. Якщо статус девайсу active, то воно поставить статус disabled та навпаки, якщо статус девайсу disabled, то воно поставить active.

```
const deleteData = (collectionName, docID) => {
    return db.collection(collectionName).doc(docID).delete();
}
```

Рис 3.6 Функція видалення даних

На рисунку 3.6 продемонстрована функція видалення девайсу. Функція deleteData отримує два аргументи:

1. collectionName – ім’я колекції з якої потрібно видалити дані;
2. docID - ID документа, який потрібно видалити;

```
const changingStatusOfManyDevices = (object) => {
    let batch = db.batch();
    object.devicesIDArray.forEach(deviceID => {
        let ref = db.collection('users-devices').doc(deviceID + '');
        if(object.currentStatus === 'active') {
            batch.update(ref, {status: 'disabled'});
            console.log('set disabled');
        } else {
            batch.update(ref, {status: 'active'});
            console.log('set active');
        }
    })
    return batch.commit()
    .then(res => console.log(res))
    .catch(err => console.log(err))
}
```

Рис 3.7 Функція зміни статусу декількох девайсів

На рис 3.7 продемонстрована функція зміни статусу декількох девайсів. Функція приймає об’єкт в якому вказано дійсний статус девайсів, там масив, який складається з ID девайсів, яким потрібно змінити статус. Якщо дійсний статус дорівнює disabled, то статус усіх девайсів, чиї ID були у масиві, зміниться на active, та навпаки, якщо дійсний статус дорівнює active, то статус усіх девайсів, чиї ID були у масиві, зміниться на disabled. Усі ці зміни відправляються одним запитом до бази даних.

3.2.3 Реалізація API для входу та реєстрації на сайті

На рисунку 3.8 продемонстровано API для аутентифікації користувача.

```
const login = app => {
    app.post('/users/login', (req, res) => {
        firestore.getData('users')
        .then(users => {
            users.forEach(user => {
                if(user.email === req.body.email && user.password === req.body.password) {
                    let token = jwt.sign({id: user.id}, 'secret', {
                        expiresIn: '24h'
                    })
                    res.status(200).send(token);
                }
            })
            res.set(401).send('there is no such user');
        })
    })
}
```

Рис. 3.8 API для аутентифікації користувача

Воно приймає POST-запит, у якому повинно міститися електронна пошта та пароль. Воно перевіряє чи існує в базі даних користувач з такою електронною поштою та паролем і якщо такий користувач існує, за допомогою бібліотеки jwt генерується токен з корисним навантаженням у вигляді ID користувача, який входить, та з терміном дії упродовж двадцяти чотирьох годин. Далі цей токен відправляється на клієнт і там він потрібен для того, щоб ідентифікувати, що саме авторизований користувач користується нашим сайтом. Цей токен майже неможливо підробити, якщо ти не знаєш секретний ключ. Якщо хакер не знає токен, то єдиний шанс його отримати, це викрасти його з комп’ютера авторизованого користувача. Якщо ж такого користувача не існує, то на клієнт відправляється відповідь зі статусом 401 – не авторизовано та повідомленням, що такого користувача не існує.

На рисунку 3.9 продемонстрована функція перевірки на унікальність електронної пошти.

Рис. 3.9 Функція перевірки на унікальність електронної пошти

Ця функція приймає аргументом електронну пошту та перевіряє, чи існує користувач з такою електронною поштою. Якщо такий користувач існує, воно відмовляє в реєстрації користувача в базі даних, та відправляє статус 400 з повідомленням що такий користувач вже існує. Якщо ж такого користувача не існує, функція нічого не відправляє.

На рисунку 3.10 продемонстровано API для реєстрації користувача.

Рис 3.10 API для реєстрації користувача

Воно приймає POST-запит у якому повинно міститися ім’я користувача, електронна пошта та пароль. Спочатку, за допомогою описаної вище функції йде перевірка на унікальність електронної пошти серед зареєстрованих користувачів. Якщо користувача з такою електронною поштою не існує, воно додає користувача з переданими даними у колекцію з користувачами та

відправляє повідомлення зі статусом 200 та повідомленням, що користувач успішно зареєстрований.

3.2.4 Реалізація API для сторінки кімнат

На рисунку 3.11 продемонстровано API для отримання доступних кімнат.

```
const roomsCatagory = app => {
    app.get('/roomsCatagory/getRooms', (req, res) => {
        console.log(req.headers.authorization)
        tokenFunctions.tokenVerification(eq.headers.authorization, res);
        firestore.getData('rooms')
        .then(data => res.send(JSON.stringify(data)));
    });
}
```

Рис. 3.11 API для отримання доступних кімнат

Це API отримує GET-запит у якому у заголовках має міститися поле Authorization у якому лежить токен. Далі за допомогою цього токена йде перевірка, чи не є він підробкою та чи не закінчився термін його валідності. Якщо з токеном все добре, то на клієнт відправляються дані з доступними кімнатами.

3.2.5 Реалізація API для сторінки девайсів

На рисунку 3.12 продемонстровано API для отримання доступних типів девайсів.

```
const sendDevicesCategory = app => {
    app.get('/devicesCatagory/getDevices', (req, res) => {
        tokenFunctions.tokenVerification(req.headers.authorization, res);
        firestore.getData('devices')
        .then(data => res.send(JSON.stringify(data)));
    });
}
```

Рис. 3.12 API для отримання доступних типів девайсів

Це API отримує GET-запит у якому у заголовках має міститися поле Authorization у якому лежить токен. Далі за допомогою цього токена йде перевірка, чи не є він підробкою та чи не закінчився термін його валідності. Якщо токен дійсний, то на клієнт відправляються данні з доступними девайсами.

На рисунку 3.13 продемонстровано функцію для додавання девайсу.

```
const firestore = require('../Firestore/firestore');

const incrementMaxID = arr => arr.reduce((acc, curr) => acc.id > curr.id ? acc : curr);

const addUserDevice = (ws, parseData) => {
    firestore.getData('users-devices')
    .then(data => {
        console.log(incrementMaxID(data).id + 1 + '')
        firestore.addData('users-devices', incrementMaxID(data).id + 1 + "", {
            'name': parseData.name,
            'deviceID': parseData.deviceID,
            'id': incrementMaxID(data).id + 1,
            'roomID': parseData.roomID,
            'status': 'disabled',
            'deviceCategoryID': parseData.deviceCategoryID
        });
        ws.send(JSON.stringify({
            'name': parseData.name,
            'deviceID': parseData.deviceID,
            'id': incrementMaxID(data).id + 1,
            'roomID': parseData.roomID,
            'status': 'disabled',
            'deviceCategoryID': parseData.deviceCategoryID
        }));
    })
    .catch(err => {
        ws.send(err);
    })
}
```

Рис. 3.13 Функція для додавання девайсу.

Ця функція приймає дані про ім’я девайсу, ID девайсу та ID кімнати, в якій цей девайс встановлений, то цей девайс додається у базу даних.

На рисунку 3.14 продемонстровано функцію зміни статусу девайсу.

```
const firestore = require('../Firestore/firestore');

const changeStatus = (ws, parseData) => {
    firestore.changeStatus(parseData.deviceID + '')
    .then(() => ws.send('Successfuly changed'))
}
```

Рис. 3.14 Функція зміни статусу девайсу.

Ця функція приймає ID девайсу, та по цьому ID змінюється статус на протилежний. Якщо нинішній статус дорівнює active, то він встановлюється у disabled, та навпаки.

На рисунку 3.15 продемонстрована функція зміни статусу декількох девайсів.

```
const changingStatusOfManyDevices = (ws, parseData) => {
    firestore.changingStatusOfManyDevices(parseData.object)
    .then(() => ws.send('Statuses changed'))
    .catch(err => ws.send(err));
}
```

Рис. 3.15 Функція зміни статусу декількох девайсів.

Ця функція визиває функцію changingStatusOfManyDevices, яка описана вище, і якщо результат виконання функції успішний, то на клієнт відправляється повідомлення: Статус змінено.

На рисунку 3.16 продемонстрована функція видалення девайсу.

```
const deleteDevice = (ws, parseData) => {
    firestore.deleteData('users-devices', parseData.deviceID + '')
    .then(() => ws.send(`${parseData.deviceID} successfuly deleted`))
}
```

Рис. 3.16 Функція видалення девайсу

Ця функція визиває функцію deleteData, яка описана вище, і якщо результат виконання функції успішний, то на клієнт відправляється повідомлення: {id девайсу} успішно видалено.

На рисунку 3.17 продемонстрована функція отримання даних про девайси.

```
const getUserDevices = ws => {
    firestore.getData('users-devices')
    .then(data => ws.send(JSON.stringify(data)))
    .catch(err => ws.send(err));
}
```

Рис 3.17 Функція отримання даних про девайси.

Ця функція визиває функцію getData, яка описана вище, і якщо результат виконання функції успішний, то на клієнт відправляються дані про девайси користувача.

На рисунку 3.18 продемонстровано API для взаємодії з даними девайсів.

```
const addUserDevice = require('./addUserDevice');
const changeStatus = require('./changeStatus');
const deleteDevice = require('./deleteDevice');
const changingStatusOfManyDevices = require('./changingStatusOfManyDevices');
const getUserDevices = require('./getUserDevices');
const usersDevices = app => {
    app.ws('/userDevices', function(ws, req) {
        getUserDevices(ws)
        ws.on('message', function(message) {
            const parseData = JSON.parse(message);
            if(parseData.type === 'addUserDevice') {
                addUserDevice(ws, parseData);
            } else if (parseData.type === 'changeStatus') {
                changeStatus(ws, parseData);
            } else if (parseData.type === 'deleteDevice') {
                deleteDevice(ws, parseData);
            } else if (parseData.type === 'changingStatusOfManyDevices') {
                changingStatusOfManyDevices(ws, parseData);
            }
        })
    })
}
```

Рис. 3.18 API для взаємодії з даними девайсів.

Алгоритм роботи продемонстровано на рисунку 3.19.

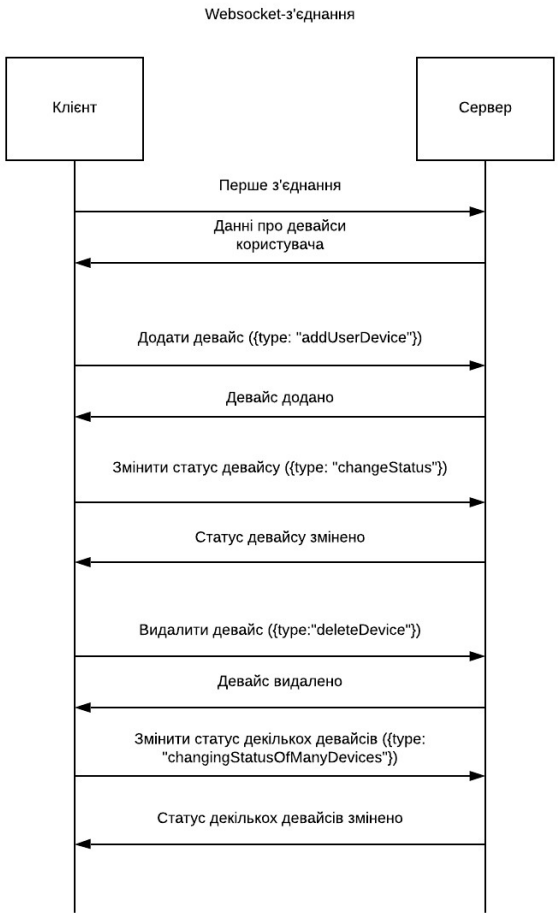


Рис. 3.19 Алгоритм роботи API для взаємодії з даними девайсів.

Це API побудовано за допомогою технології Websocket. Якщо відбувається з’єднання з сервером, то на клієнт відправляються масив з девайсами користувача. Якщо з клієнта приходить повідомлення з об’єктом, який містить в собі type: addUserDevice, та дані про ім’я девайсу, ID девайсу та ID кімнати, в якій цей девайс встановлений, то цей девайс додасться у базу даних. Якщо ж прийде повідомлення з об’єктом, який містить в собі type: changeStatus, та ID девайсу, якому потрібно змінити статус, то за допомогою функцій описаних вище виконується зміна статусу потрібного девайсу. У відповідь відправляється повідомлення, що статус успішно змінено. Якщо ж прийде повідомлення з об’єктом, який містить в собі type: deleteDevice, та ID девайсу, яке потрібно видалити, то за допомогою функцій описаних вище виконується видалення потрібного девайсу. У відповідь відправляється повідомлення, що девайс видалено. Якщо ж прийде повідомлення з об’єктом, який містить в собі type: changingStatusOfManyDevices, та об’єкт, який містить в собі теперішній статус, та масив, який містить в собі ID девайсів, яким потрібно змінити статус. Якщо теперішний статус дорівнює active, то статус усіх девайсів, чиї ID містяться в масиві, установиться в disabled, та навпаки.

ВИСНОВКИ ДО РОЗДІЛУ 3

У даному розділі був проведений аналіз технологій розробки серверної частини, а також аналіз додаткових бібліотек, які допомогли вирішити поставлену задачу. Враховуючи усі переваги та недоліки було обрано мову програмування Javascript та бібліотеку Node.js для розробки поставленої задачі.

Реалізацію серверної частини було розбито на таки етапи:

- Реалізація функцій для використання JWT;
- Реалізація функцій для використання Firestore;
- Реалізація API для входу та реєстрації;
- Реалізація API для сторінки кімнат;
- Реалізація API для сторінки девайсів;

У результаті була побудована серверна частина для системи безпеки розумного дому з використанням web-socket.

Були наведені рисунки з кодом та опис алгоритму виконання цього коду.

ВИСНОВКИ

Розробка даного дипломного проекту була присвячена дослідженню існуючих варіантів систем розумного дому.

У ході роботи було розглянуто вже існуючи рішення, які реалізують системи розумного дому. На основі цих досліджень було складено порівняльну характеристику у якій було прораховано всі недоліки та переваги існуючих систем.

На основі минулих досліджень було визначено основні вимоги до програмного продукту, а також функції, які можуть бути корисні користувачу. Також було проведено проектування бази даних та побудовано відповідну базу даних.

Зважаючи на вище задані вимоги був проведений аналіз існуючих технологій та систем для реалізації системи. Ця серверна частина може бути використана не тільки для веб додатку, а й для розробки мобільного додатку.

Також було проведено аналіз, які технології краще використовувати, та за результатами аналізу було обрано мову Javasctipt та бібліотеку Node.js.

Реалізація відбувалася у 5 етапів.

СПИСОК ВИКОРИСТОНАЇ ЛІТЕРАТУРИ

1. Websocket [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/websocket
2. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Node.js
3. Common.js [Електронний ресурс] – Режим доступу до ресурсу: http://largescalejs.ru/commonjs-modules/
4. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://artjoker.ua/ru/blog/v-chem-preimushchestva-nodejs/
5. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://devzone.org.ua/post/nodejs-abo-java-produktivnist-resursi-upravlinnya-potokami-populyarnist-i-osobistiy-dosvid
6. V8 [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/V8_(%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA_JavaScript)
7. Express.js [Електронний ресурс] – Режим доступу до ресурсу: *https://ru.wikipedia.org/wiki/Express.js*
8. The JWT Handbook Sebastian E. Peyrott, Auth0 Inc.
9. JWT [Електронний ресурс] – Режим доступу до ресурсу: *https://codex.so/jwt*
10. Node.js [Електронний ресурс] – Режим доступу до ресурсу: *https://senior.ua/articles/pochemu-node-js-osobennosti-i-preimuschestva*
11. Java [Електронний ресурс] – Режим доступу до ресурсу: *http://www.ccc.ru/magazine/depot/99_13/0801.htm*
12. Python чи Node.js [Електронний ресурс] – Режим доступу до ресурсу: *https://nuancesprog.ru/p/4281/*
13. Розумний будинок [Електронний ресурс] – Режим доступу до ресурсу: *https://allcosmoshop.ru/uk/otoplenie/the-latest-technology-smart-home-smart-home-with-their-own-hands-a-scheme-and-equipment/*

14. Розумний будинок [Електронний ресурс] – Режим доступу до ресурсу: *https://vencon.ua/ua/articles/reyting-sistem-umnyy-dom-po-proizvoditelyam*
15. База даних [Електронний ресурс] – Режим доступу до ресурсу: *https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85*
16. Бази даних [Електронний ресурс] – Режим доступу до ресурсу *https://www.oracle.com/database/what-is-database.html*
17. Розумний будинок [Електронний ресурс] – Режим доступу до ресурсу *https://domsireni.ru/uk/otoplenie/a-smart-house-with-your-own-hands-smart-home-system-highly-intelligent-housing/*

ДОДАТОК 1

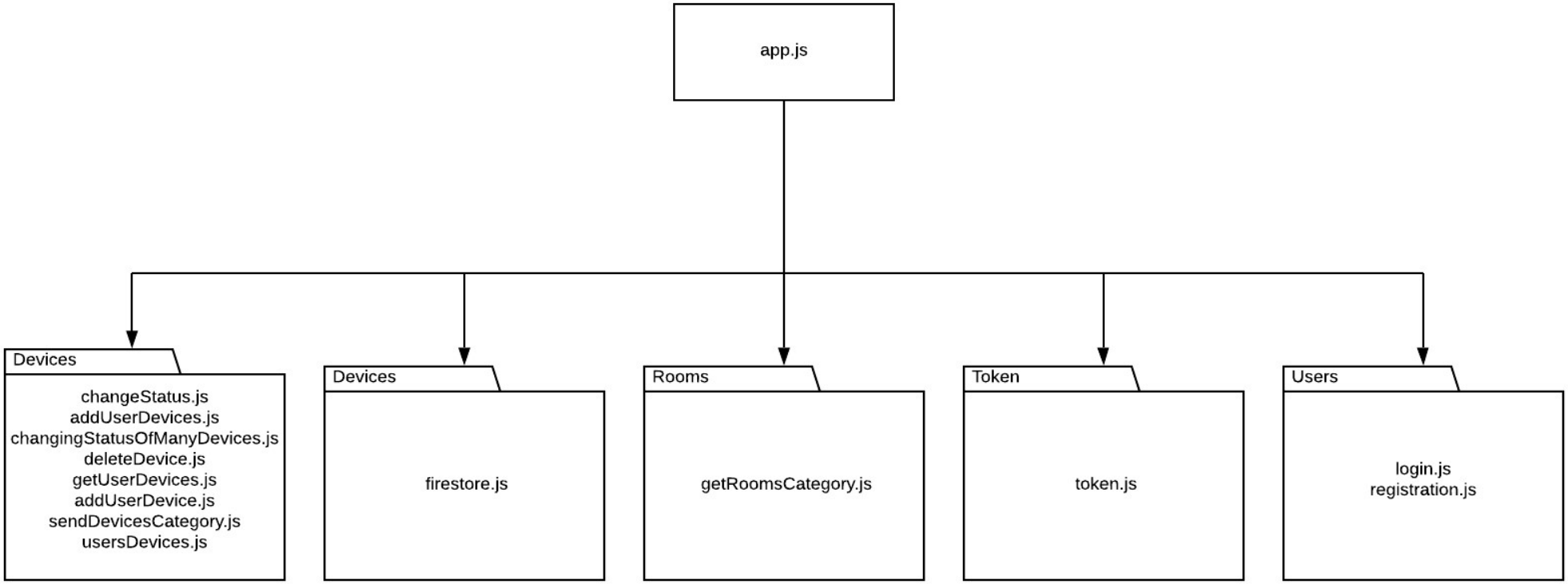
Система безпеки розумного дому з використанням web-socket
(серверна частина)

Схема структурна

ІАЛЦ.467100.004 Д1

Аркушів 1

Київ – 2020 р.



					ІАЛЦ.467100.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Лахуров А. І.			Система безпеки розумного дому з використанням web-socket (серверна частина)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Стешин В.В.				
Реценз.						
Н. Контр.		Сімоненко В.П.			Схема структурна	НТУУ «КПІ», ФІОТ, ІО-63
Затв.		Стіренко С.Г.				

ДОДАТОК 2

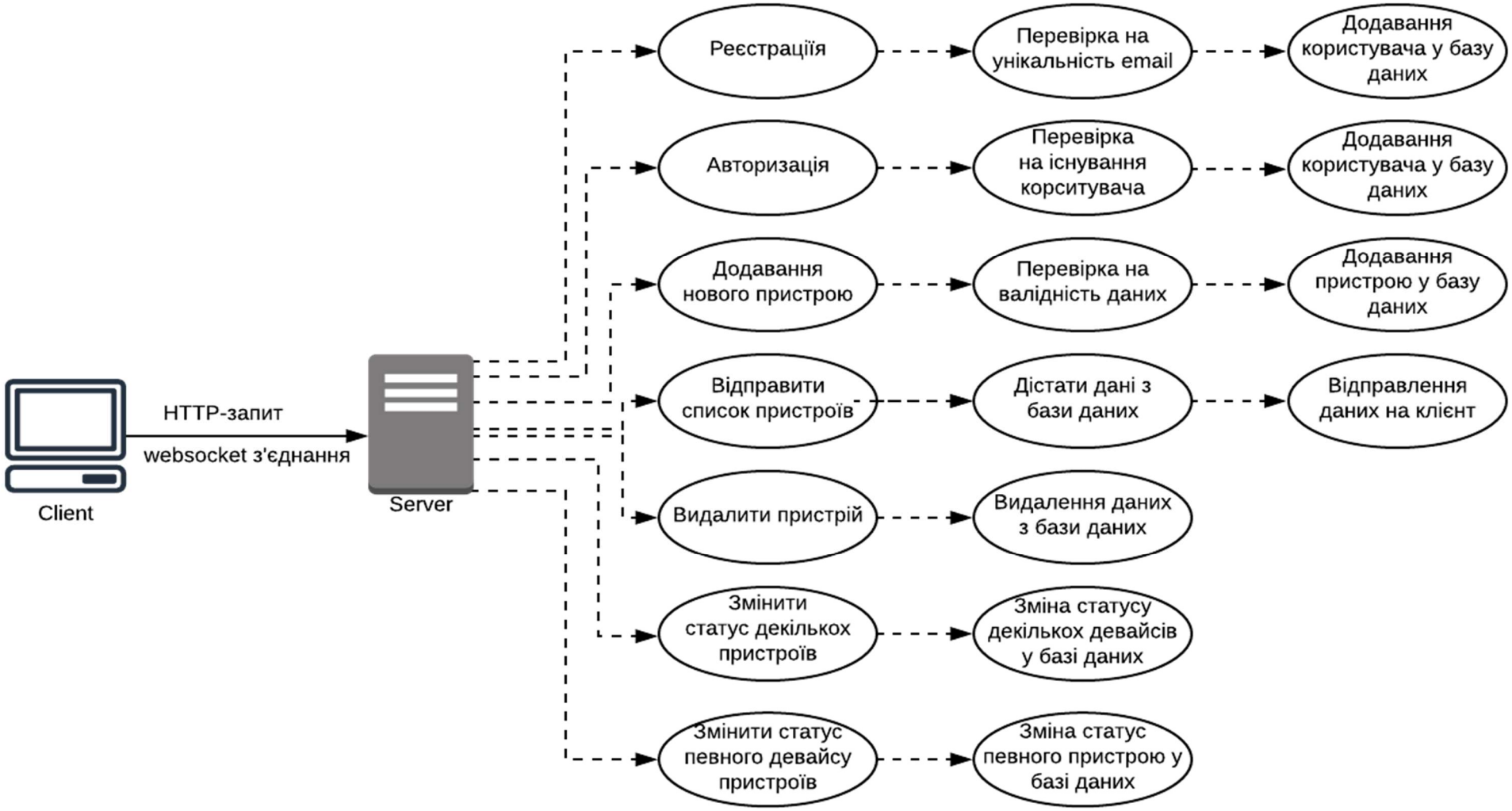
Система безпеки розумного дому з використанням web-socket
(серверна частина)

Схема функціональна

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ – 2020



					ІАЛЦ.467100.005 Д2
Зм.	Арк.	№ докум.	Підпис	Дата	
Розробив		Лахуров А. І.			Система безпеки розумного дому з використанням web-socket (серверна частина)
Перевірив		Стешин В.В.			
Реценз.					
Н. Контр.		Сімоненко В.П.			Схема функціональна
Затв.		Стіренко С.Г.			

Літ.	Аркуш	Аркушів
	1	1

НТУУ «КПІ», ФІОТ, ІО-63

ДОДАТОК 3

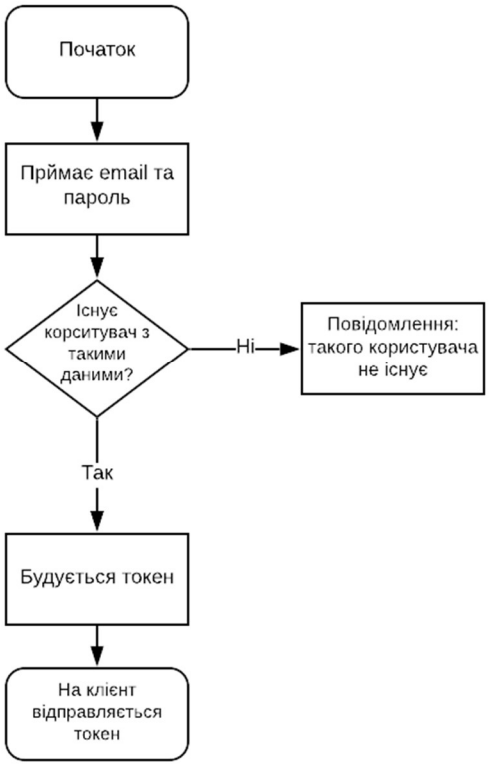
Система безпеки розумного дому з використанням web-socket
(серверна частина)

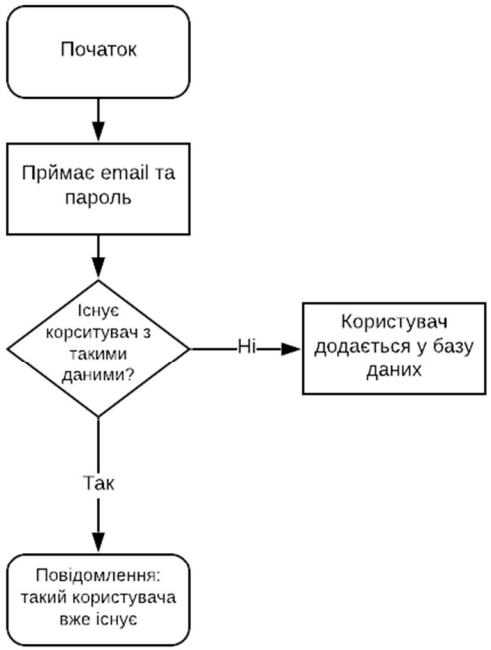
Схема принципова

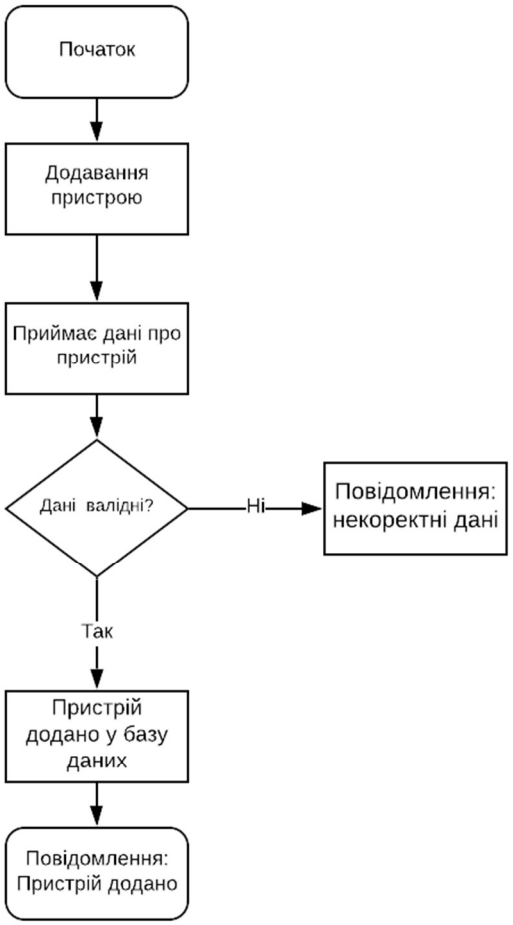
ІАЛЦ.467100.006 Д3

Аркушів 1

Київ – 2020







					ІАЛЦ.467100.006 Д3	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Лахуров А. І.*			*Система безпеки розумного дому з використанням web-socket (серверна частина)*	*Літ.* / *Аркуш* / *Аркушів*
Перевірив		*Стешин В.В.*				*1* / *1*
Реценз.						
Н. Контр.		*Сімоненко В.П.*			*Схема принципова*	*НТУУ «КПІ», ФІОТ, ІО-63*
Затв.		*Стіренко С.Г.*				